JPRS-UCC-90-002
27 MARCH 1990

FOREIGN
BROADCAST
INFORMATION
SERVICE

JPRS Report

Science & Technology

USSR: Computers

JPRS-UCC-90-002
27 MARCH 1990

SCIENCE & TECHNOLOGY

USSR: COMPUTERS

CONTENTS

ABSTRACTS

18630180C Novosibirsk AVTOMETRIYA in Russian No 6, Nov-Dec 88 (manuscript received) pp 120-122 [Source-supplied Russian abstracts of articles published in AVTOMETRIYA, No 6, 1988]

[Text]

UDC 621.372.542
V.A. Ivanov, "Algorithms Generating a Series of Nonstationary Images Based on a Mosaic Model," AVTOMETRIYA, No 6, 1988.
The objective is to simulate a series of nonstationary images needed for simulation of a broad battery of image types and to validate and test algorithms processing image series. A mosaic model is validated as a means for generation of the microstructures of images simulated. Algorithms are developed which synthesize images on a mosaic with random boundaries, move mosaic unit cells in coordinates and vary brightness to form a series of nonstationary images. A software package based on this algorithm has been created which is run on an E-79 computer.

UDC 519.24
V.G. Alekseyev, "Choice of Evaluation Parameters for a Regression Curve by a Crossverification Methods," AVTOMETRIYA, No 6, 1988.
A crossverification method is described which is used for nonparametric evaluation of a regression curve that depends on two functional and two numeric parameters.

UDC 621.391
Yu.A. Kapitonov, Yu.I. Palagin and A.S. Shalygin, "Reconstructing the Distribution Density of Random Fields and Processes From Experimental Data by Using Kernel Estimates," AVTOMETRIYA, No 6, 1988.
Rosenblatt-Parsen kernel estimates are employed to reconstruct distribution densities of random fields and processes. The suitability of an asymptotic theory of independent samples is investigated. The effect of correlation characteristics of fields and processing parameters on estimate errors is

analyzed. A comparison with the method of histograms is made and illustrations are given.

UDC 621.391.2
A.S. Danyayev and V.N. Prokofyev, "Invariant Algorithms of Automatic Detection of 'Signal Trajectories,'" AVTOMETRIYA, No 6, 1988.
Detection of a signal in a specified frequency-time (or space-time) domain is studied under the assumption that the unknown "location" of the signal in the domain need not be constant during the time of analysis (it forms a trajectory) and that the signal and noise levels not known a priori can also vary in the analysis domain. Invariant algorithms are proposed which detect and identify the location of the "signal trajectory," which can be implemented in automatic devices; the efficiency of the algorithms is noted.

UDC 621.397.2:519.685
I.M. Bokshteyn, "Component Transformation Method Used to Reduce Redundancy in Color Images," AVTOMETRIYA, No 6, 1988.
Possible applications of the component transformation method with interpolation in readings are evaluated as a way to reduce redundancy of multigradational color images. A version is suggested which ensures a high degree of compression (up to 2.62 bits/reading) with a high reconstruction quality. An optimal distribution of the number of digitization levels with respect to the components is found. The results of a simulation are described which tested the capabilities of the method on processing of real color images. The test confirmed the efficiency of the component transformation method.

UDC 681.327.68:778.38
V.A. Dombrovskiy, S.A. Dombrovskiy and Ye.F. Pen, "Reliability of Information Reading in a Holographic Memory Channel With Constant Parameters," AVTOMETRIYA, No 6, 1988.
Calculated data reading reliability in holographic memory based on an assumption of normal distribution and a generalized Rayleigh-Rice distribution of image powers "1" and "0" is compared with the reliability measured experimentally. The reading reliability is investigated as a function of the photometric characteristics and photoscanning regime.

UDC 517.44:621.391.1:681.3.012
O.Ye. Baklanova, M.V. Zyuzin and A.V. Lyulyakov, "Implementation of Parallel Digital-Filtering Algorithms on the PS-2000 Multiprocessor Computation Complex," AVTOMETRIYA, No 6, 1988.
Algorithms for the digital filtering of signals and images implemented on a PS-2000 multiprocessor computation complex are described. Calculations demonstrating the efficiency of the processing of experimental data are reported.

UDC 681.3.06
A.M. Kovalev and Yu.V. Tarasov, "Texture on Arbitrarily Oriented Flat Surfaces," AVTOMETRIYA, No 6, 1988.
Mapping a texture file onto an arbitrarily oriented plane in a three-dimensional space is studied. A method of mapping is proposed that is suitable for hardware implementation and can be used in multiprocessor real-time computer graphics systems.

UDC 535.853.23:578.08
L.A. Avramov, V.N. Verkhoturov, V.V. Gorokhov, A.I. Komarov, B.P. Korvatovskiy, A.A. Lazarev, V.Z. Pashchenko, A.Ya. Pikulenko and A.B. Rubin, "Computerized Absorption Spectrometer for Kinetic and Spectral Studies in the Picosecond Time Range," AVTOMETRIYA, No 6, 1988.
A computerized absorption spectrometer is described designed for kinetic and spectral studies in the picosecond time range. The software for the experimental installation is described. The capabilities of the unit have been demonstrated in a test measuring the variation kinetics of optical absorption with a wavelength 652 nm.

UDC 681.3:519.246.8
D.N. Zabiyaka, A.A. Predtechenskiy and A.I. Chernykh, "Spectrum Analyzer Based on an Elektronika 60 Microcomputer," AVTOMETRIYA, No 6, 1988.
A narrow-band digital spectrum analyzer is described for signals in the sonic-frequency range (up to 16 kHz) and the infrasonic range. The analyzer is based on the DVK-2 and Elektronika 60M microcomputers without any structural modifications. A software implementation of the periodogram method and of related spectrum-analysis algorithm is capable of real-time signal processing in a band of up to 250 Hz.

UDC 621.39.1:621.378:532.57
V.A. Zhmud, "Servoprocessor for LDIS [not further identified] Signals in Backscattering Mode," AVTOMETRIYA, No 6, 1988.
A new system is described which processes LDIS signals of a hydraulic flow in backscattering mode. The core element of the system is a servoprocessor featuring high accuracy, operation speed and noise immunity due to a new memory circuit, a combination of monitor and frequency signal quality criteria and introduction of an additional acquisition criterion.

UDC 681.335.2
L.K. Samoylov and G.I. Tkachenko, "Data Collection Program Generation in Information and Measurement Systems," AVTOMETRIYA, No 6, 1988.
Various algorithms forming data collection programs and their shortcomings analyzed. New algorithms are described which ensure a higher efficiency in utilizing the communication channel throughput and allow placing the appropriate number of synchronization pulses in a cycle in neighboring intervals. A block diagram of the computer algorithm is given which forms the effective data collection program for an arbitrary set of desired readout

frequencies under the assumption that synchronization pulses are placed in neighboring intervals.

UDC 535.4:778.38
I.G. Palchikova, "Kinoform Conoid Axicons," AVTOMETRIYA, No 6, 1988.
A new method is proposed for calculating the characteristics of kinoform axicons which can produce a desired intensity distribution along the focal segment. Experimental studies of axicon caustics are reported.

UDC 681.327.68:778.38
V.K. Yerokhovets, "Estimating the Tolerances of Linear Positioning of Components in Documentary Graphic Memories," AVTOMETRIYA, No 6, 1988.
In general mathematical terms, the tolerances of linear positioning of the components of a graphic memory are estimated (a transparency with initial data, point light sources in reference and reading channels of the memory) in terms of sharpness and accuracy of the format reproduced.

UDC 621.396:535.8
S.Yu. Bondartsev, N.A. Yessikina and A.P. Lavrov, "Optical Processors With Charge-Coupled Photodetectors," AVTOMETRIYA, No 6, 1988.
Optical processors with scanning CCD photodetectors are discussed. The processors are shown to be a promising technology for analysis of various types of radio signals. A study of an optical correlator with a reference mask has been conducted. It generates crosscorrelation functions of a video signal and a library of reference signals written in the mask. The correlator is shown to be a highly productive optical processor which accomplishes vector-matrix multiplication. The paper also describes an acoustooptical device processing signals of pulsars (acoustooptical dispersion compensator); with the aid of a scanning CCD photodetector pulsar, signals are condensed.

UDC 681.7.013.82
V.I. Kozik, A.N. Oparin and O.I. Potaturkin, "A Study of the Characteristics of a Holographic Correlation Conjugated With a Video Processor," AVTOMETRIYA, No 6, 1988.
An optoelectronic system is described which consists of a holographic intensity correlator [GKI] with a quasimonochromatic CRT and a universal digital video processor controlled by a microcomputer and coupled with the correlator in input and output. The influence is studied of the parameters of the optical GKI system and the quasimonochromatic CRT upon the resolution and transmittance of the correlator. The recognition efficiency of real objects with digital image preprocessing algorithms is evaluated. Results of an experimental study are reported.

UDC 621.391
V.G. Getmanov, "Reducing the Time of Solution of Discrete Trigonometric Approximations," AVTOMETRIYA, No 6, 1988.

Approximation of narrow-band processes by a modeling sinusoidal function is studied. The residual sum of squares in the frequency region is registered by using a discrete Fourier transform of the initial process and the model function. This helps reduce substantially the model parameter calculation time compared with approximations in the time region.

UDC 381.3.06
A.A. Yantimirov, "Software for Raster Graphics Devices," AVTOMETRIYA, No 6, 1988.
Algorithms of a standardized basic software package that implements the graphics function independent of the type of hardware are described. The program draws polygons and parts of circles, filling them with patterns.

Soviet Personal Computers

907G0051A Moscow NOVOYE V ZHIZNI, NAUKA, TEKHNIKE: VYCHISLITELNAYA TEKHNIKA I EYE PRIMENENIE in Russian No 11, Nov 1989 pp 22-29

[Full translation of article "Personal Computers" by A. A. Stanishevskiy]

[Text] The first personal computer, i.e., the first personal microcomputer, appeared in 1976 when the first "Apple" computer was manufactured. The birth of the first microcomputer was also accompanied by the development of a new terminology which is often borrowed from the "big brothers" — mainframe and minicomputers, and this terminology has to some degree followed the development of the new "world" of personal computers.

One journal for beginner users of computer technology has explained that since the central processor is based on an integrated circuit — a microprocessor — the computer itself is called a microcomputer.

An article in *Literaturnaya Gazeta* on this subject has noted quite accurately that in order to be a personal computer a computer must be bought in a store, brought home, and used independently.

However, it would be more accurate to speak of microprocessor technology rather than the microcomputer since it is microprocessor technology that is based on microprocessors. Since microprocessors are installed and used not only in computer technology but also in a wide variety of domestic equipment and industrial engineering a microprocessor-based computer installed in such equipment will be called an embedded microcomputer. Such a microcomputer is characterized by a design whereby both its tasks and functions are predetermined and are limited to its future application. All required software is "woven" in advance in the RAM of this computer whose tasks involve performing any of a variety of industrial processes.

A microcomputer that is used for domestic purposes, for teaching students, for games, and which has a limited RAM that ordinarily does not exceed 64 Kbytes without advance peripherals and is contained within a single unit plus a keyboard will be referred to as a home computer. The "Elektronika BK0010", "Mikrosha" and "Radio RK86" microcomputers are the domestic represents of such computers.

A microcomputer whose RAM begins at 512 Kbytes and can be expanded to 1 Mbyte or more and which has advance peripherals: floppy disk memories, "Winchester" hard disks (less than 10 Mbytes) a printer and any other advance peripheral depending on the field of application will be called a professional personal computer.

Of course the definitions of microcomputers given here are rather arbitrary since a home computer can also be used to run moderately complex industrial tasks, while a professional personal computer can be used for training.

The "Elektronika BK0010" domestic personal computer.

The "Elektronika BK0010" microcomputer became the first domestic personal computer that was in fact available for purchase in stores. Its popularity was reflected by the fact that it could not be kept on store shelves and many informal societies of users of the "Elektronika BK0010" microcomputers were formed.

Of course one of the reasons for the polynomial of the "Elektronika BK0010" computer was that it was the only computer available in the stores and it was relatively "inexpensive": approximately 600 rubles. Another reason for the popularity was its simple design: the entire computer was housed in a single unit with a keyboard and its dimensions were such that it could easily be fit in a briefcase.

The random access memory in the "Elektronika BK0010" computer was 16 Kbytes with an additional 16 Kbytes used for the screen memory. Several operating modes could be used and the RAM could be increased to 28 Kbytes to accommodate these.

Such a RAM capacity is, of course, quite small, although it does permit execution of simple programs that can be written in BASIC. It is also possible to use finished programs by inputting these to the RAM from magnetic tape. A variety of graphs and figures can be generated on the "Elektronika BK0010". Nonetheless the RAM capacity is quite small.

Therefore in future models of the computer — the "Elektronika BK0011" and the "Elektronika BK0100" — the RAM capacity will be increased to 128-256 Kbytes.

The "Agat" personal computer

The "Agat" personal computer is another representative of domestic computers used for teaching purposes. A modular design principle was used in constructing this model; this design makes it possible to use various computer configurations.

The "Agat" computer consists of a system unit, a keyboard, a video monitor, and a dot-matrix printer.

The 6502 microprocessor manufactured by MOS technology is used in the "Agat" computer as the primary element in the central processor. The microprocessor utilizes 8-bit data. Its clock frequency is 1 MHz with an address bus word size of 16 bits which permits addressing of a 64 Kbyte RAM.

The RAM capacity in the "Agat" computer is 32 Kbytes and is expandable to 256 Kbytes. The ROM capacity of the computer is 32 Kbytes.

The system unit of the computer contains the mother board, the auxiliary modules, the power supply, the floppy disk memories, and auxiliary expander units. A maximum of seven modules can fit in this unit.

The internal interface of the "Agat" computer is used to interconnect the modules. The internal system interface of the computer is implemented as a set of 7 60-pin connectors on the mother board. The internal system interface is divided into a data bus — 8 lines, an address bus — 16 lines, and a control and timing bus.

The mother board contains the central process module, the RAM module, the floppy disk memory controller module, and the serial and parallel interface modules.

The "Agat" computer RAM is divided into 256 byte pages. The instruction set of the central processor is determined by the 6502 microprocessor which is used to maintain program compatibility with the Apple II and TRS 80 foreign personal computers.

The monitor (display) of the "Agat" computer is based on a "Yunost' Ts404" domestic television receiver. The unit contains an interface to the "Agat" computer and bypasses the RF circuit. The video monitor makes it possible to generate the following images:

- output of graphics data: black and white image 256 × 256 pixels and a 128 × 128 and 64 × 64 pixel color image;

- for alphanumeric data output: black and white image of 30 × 64 and a color image of 32 × 32 characters.

Both a domestic tape recorder for cassette programs and ES5088 floppy disk memories manufactured by the Peoples' Republic of Bulgaria and ES5089 manufactured by the German Democratic Republic are used as the external memories in the "Agat" computer. The floppy disks are 133 mm in diameter and the write format accommodates 140 Kbytes of data per magnetic disk.

The system employs a D100 dot matrix printer which produces a fine Russian font and Latin font. The DZM180 and EPSON-CENTRONIX printers can also be used.

The computer keyboard contains 26 Latin and 31 Russian Characters, 10 digit keys and 23 control keys.

The computer utilizes the following software: the "Agat" DOS operating system, and a BASIC language interpreter which is the Microsoft dialect of BASIC.

The program monitor is stored in the ROM of the "Agat" computer. The system monitor supports the set up of the central processor operating modes ("cold start") and initiation of the software routines, swapping with the magnetic tape memories (domestic tape recorder), control of keyboard commands, control of data input from magnetic tape, acquisition and loading of programs from the floppy disk. If no floppy disk is available the system goes to a "system monitor interaction" mode.

The "Agat" DOS operating system is designed to generate, track, and delete user data sets (files) stored on floppy disks. The DOS can use three types of files:
A - BASIC program;
B - binary program;
T - text program.

The "Agat" computer operating system can support up to 10 floppy disk memories (of the ES5089 type).

The operating system and software make the "Agat" computer an excellent tool for teaching. The "Agat" computer can also be used for simple data retrieval purposes.

DVK: the interactive computer system

The DVK-2 was one of the first domestic personal computers. This 16-bit computer was based on the K1801 microprocessor. The DVK computers were compatible in architecture to the SM3 and SM4 minicomputers as well as the PDP/11 minicomputers of foreign manufacture.

The primary specifications of the DVK personal computer are as follows:

data word size - 16 bits;
basic RAM - 64 kbytes.

The DVK contains two disk drives for 130 mm floppy disks with a total information capacity of 2 × 216 Kbytes. The larger 208 mm disk drives can also be used in the DVK since floppy magnetic disks of this diameter have been used in the SM4 minicomputer.

The DVK computer includes a separate power supply, and a system unit which contains the central processor board, the RAM, the keyboard, and the display.

In the DVK-2 model an alphanumeric display (a character display) is used and the keyboard contains 64 keys; in the DVK-3 model a graphics display with a 600 × 320 pixel resolution is used together with a completely new keyboard expanded to 115 keys and is easy to use.

The DVK-3 computer expands the DVK family; this computer has a graphics monochromatic display and a new keyboard which has substantially expanded its possible applications. There is also a DVK-4 model with a color display.

What has been responsible for the broad popularity of the DVK models in practice and has at the same time limited their use? The answer must be sought not only in the technical design of the DVK series, although this is important, but also in the "software" product, which accompanies it.

As noted above the DVK computer models have an architecture that is compatible with the SM4 series minicomputers. Initially this permitted the use of mathematics designed for the SM computer. The DVK series operating system is the DVK OS which at the same time is an abbreviated version of the RAFOS operating system. Some utilize the RT-11 operating system in its shortened version in place of the DVK operating system. By itself using an identical operating system on different types of machines is a positive aspect and supports compatibility and transferability of the programs.

However, the programs designed for the SM microcomputer largely require external hard disk memories, i.e., a significantly larger external memory capacity and a larger RAM capacity. Therefore for the SM computer the RAM capacity is at least 128 Kbytes.

By itself the nucleus of the operating system occupies a comparatively small volume of the RAM, although the DVK operating system is designed so that the unnecessary modules in the operating system are loaded when needed which is rather difficult with 64 Kbytes of DVK RAM which must store the nucleus of the operating system and the user programs simultaneous.

Large program packages that require substantial RAM space cannot be used on the DVK computer. When an external memory is used the nucleus and the utilities of the operating system occupy virtually the entire space on one diskette (this also makes it impossible to use all the capabilities of the operating system), while nearly all the space on the second diskette is used for holding the language to which the editor translates, the programming language translator, the linkage editor, and the file of object modules of standard functions. It is difficult to fit this entire set on one diskette. It is also necessary to rearrange the diskettes as space must be found for storing new and existing programs.

FORTRAN, PASCAL, and BASIC are among the programming languages that have found broad application on the DVK computer. The DVK employs an RDB-microrelational data base. LISP and PROLOG artificial intelligence packages have been developed together with graphics packages for PASCAL and PROLOG operation. All these capabilities permit use of the DVK computer for a broad range of applications.

"Elektronika MS 0585" personal computer system

The designers of the "Elektronika MS 0585" personal computer designed the unit for operation in automated manufacturing management systems, scientific and engineering automation systems for producing electronic equipment and for processing industrial economic and statistical data.

This computer system has, like all modern personal computers, a modular design principle. All its functional units are complete devices (modules) connected by means of a communications cable or a system bus.

All functional units of the "Elektronika MS 0585" computer are contained within the single housing of the system unit. The system unit has a simple internal architecture. It contains four complete functional units. This includes a power supply with a fan, a hard disk memory, a double floppy disk memory, and an auxiliary module set. All four units are connected to the same metallic chassis. The system board is also on the chassis and can be removed using a lever (the board can be extracted from the side of the system unit.

The layout of all components provides easy access for simple computer assembly and disassembly. An additional "Winchester" memory can be used in place of the twin floppy disk memories.

One feature of the "Elektronika MS 0585" computer is the system module which uses a separate board. The system module contains: the central processor, the RAM dynamic distribution control, the RAM (512 Kbytes), the ROM which employs page organization (4 pages of 4 Kbytes each), the keyboard controller, the peripheral controller, and the 50 byte RAM timer. All devices in the system module are independent and interconnected by a bidirectional internal bus.

The system bus is used to establish a high speed link between the central processor and the peripherals. The system bus is connected to the "Winchester" hard disk memory controller, the floppy disk controller, the video controller, and the video controller memory expander.

The screen memory is 32 Kbytes and can be expanded to 96 Kbytes by using the expander. The floppy disk memory employs 133 mm diskettes and the recording format permits storage of up to 400 Kbytes. The information capacity of the hard disk is 5 or 10 Mbytes depending on configuration.

The "Winchester" hard disk makes the "Elektronika MS 0585" unit a professional computer. The RAM capacity is expandable to 4 Mbytes.

The 16-bit address bus used in the computer makes it impossible to address more than 64 Kbytes of RAM. However, the memory dispatcher can be used to address up to 4 Mbytes and also supports simultaneous execution of several processes, i.e., multiprogram operation. The memory dispatcher permits execution of tasks regardless of their location in the RAM. Each task is run in its own virtual space of 64 Kbytes while the memory dispatcher provides a reference between the logic addresses and the physical addresses.

The computer central processor is based on the K1811 microprocessor set. The "Elektronika MS 6105" black and white graphics monitor is used as the display. The graphics appear on screen in dimensions of 640 × 320 pixels in 16 gray scale gradations.

The system bus supports two data swapping modes: program swapping and program interrupt swapping. The program data swapping mode supports data transmission by request and under program control. The interrupt data swapping mode runs programs via a peripheral program. In this mode the central processor interrupts the execution of the present program, and stores its present state in order to handle the device from which the interrupt has been sent.

The power switch is located on the front panel of the system unit which is easy to use, although there is no "hot" start key, i.e., there is no way to reload the operating system without turning the computer off.

The "Elektronika MS 0585" computer software includes the PROS or FODOS operating system.

The PROS operating system is convenient for the nonprofessional user since it interacts with the user through a menu system. PROS is used for bootstrapping and execution of applied service programs, supports the operation of the peripherals and the entire computer as well as multiprogram operation. PROS includes a BASIC language interpreter and a text editor.

The test software includes test programs for the primary computer components. The computer is tested when the operating system is loaded. A cross-section image of the computer appears on the monitor screen and when one of the units is shaded this indicates that the unit has failed.

The FODOS operating system supports all the computer devices. FODOS contains ASSEMBLER and FORTRAN program languages.

Since the instruction set of the central processor of the "Elektronika MS 0585" is identical to that of the SM4 and SM1420 minicomputers, the same software can be run on the former as on the latter. In principle the "Elektronika MS 0585" is an SM1420. Hence the spectrum of the software is quite broad. This includes the RT11, RSX 11M, OS, RAFOS, RV, and DEMOS operating systems and the applied program products controlled by these operating systems.

For example, the DEMOS operating system transforms the "Elektronika MS 0585" personal computer into a powerful tool which utilizes the C, and PASCAL high level programming languages as well as an advanced catalog system and the RUBEN high level relational data base. The latter interacts with other computers by means of the KERMIT package.

The "Neuron 66" personal computer

The designers gave this somewhat poetic name to their personal computer. It should be pointed out immediately that this computer has no relation to neural networks or computers based on the principles of neural nets.

Nonetheless the "Neuron 66" can claim the name of "personal computer". Like all modern computers it is based on a modular design principle, and has a modern operating system. The "Neuron 66" computer is designed to automate the collection, processing, and documentation of information as an intelligent controller in computerized instrument systems for controlling radio instruments. And, of course, the "Neuron 66" is designed for automation of individual efforts in design and scientific and technical activities.

The modular design principle where each module is a complete device makes it possible to easily construct different versions of the "Neuron 66" personal computer system. Its base units include.

- the controller (system unit) which contains the system microprocessor board, the RAM and ROM, and the power supply;

- the floppy disk memory;

- the "Winchester" hard disk memory;

- the video monitor;

- the keyboard.

A printer and a plotter can be connected to the controller depending on the configuration.

A KP 1810 VM86 16-bit microprocessor is used to construct a central processor as the base element. The RAM capacity can vary from 256 to 520 Kbytes. The recording format yields a 640 Kbyte floppy disk capacity. The data capacity of the hard disk is 5 Mbytes.

The system employs the same "Elektronika MS 6105" black and white display as the DVK 3M, "Elektronika MS 0585" and ES1840 personal computers. The display produces a 640 × 200 pixel mode in the graphics mode and generates 25 lines of 80 characters each in the alphanumeric mode.

The computer utilizes a C2 interface. The front panel of the controller has a "reset" button which makes it possible to reload the operating system in the event of any difficulties (a computer "hang-up") without shutting the computer down.

The software for the "Neuron 66" personal computer consists of system software, applied software, and test software.

The selection of system software for the "Neuron 66" is rather large. Such software includes the "Neuron DOS1", the "Neuron DOS2" systems, and there is also program compatibility with the popular CPM/86 and MSDOS operating systems. It is clear from a comparison of the instruction sets of the operating systems that the operating systems of the "Neuron 66" computer represents their analogs. This makes it possible to use the same software as used with MSDOS and CPM/86. This would all seem fine although certain difficulties remain.

The floppy disk write format used in MSDOS is a 360 Kbyte format which was also well understood by the designers of the ES1840 personal computer. The incompatibility of the floppy disk format causes certain difficulties, unnecessary time losses in transferring information from one operating system to another, and from one computer to another. The 512 Kbyte RAM and even more so the 260 Kbyte RAM cannot handle many modern applied program packages. The standard minimum RAM capacity today is 640 Kbytes and the designers of modern software design for this capacity.

It is also to use the capabilities provided by the "Neuron 66" personal computer which includes the BASIC and PASCAL high level programming languages.

The "Iskra 1030" personal professional computer

The "Iskra 1030" is one additional personal computer that can fill the gap where a minicomputer or mainframe computer is either not suitable or is too expensive.

The designers of the "Iskra 1030" personal computer designed the unit for automatic execution of a wide range of economic, administrative and management routines.

The modular design principle was also used in the "Iskra 1030" personal computer, thereby simplifying its outfitting and the connection to various auxiliaries.

The basic setup of the "Iskra 1030" computer includes five functional units (modules):P

- processor module (system unit);

- data display module (display);

- keyboard module;

- printer;

- distributor filter unit.

We are already familiar with the first four modules. The distributor filter unit is designed to filter the standard supply voltage and the voltage to the first four modules.

The system unit (processor module) contains virtually all primary computer components within a single housing:

- the system board containing the microprocessor;

- the RAM and data display module controllers;

- the 133 mm floppy disk controllers;

- the "Winchester" hard disk controller;

- the data transmission system adapter.

Aside from these sections the system unit contains the floppy disk drives, the "Winchester" hard disk drive and the power supply.

The system unit therefore is rather large in size: 485 × 455 × 200 and it was then decided to install all primary computer components within a single unit which was not a design choice for the designers of the ES1840 and "Neuron 66" computers.

The K1810 VM86 microprocessor used in the "Iskra 1030" operates at a clock frequency of 4.77 MHz with 16-bit data while a 20-bit address bus is used for addressing. The "Iskra 1030" RAM has a 512 Kbyte capacity. It is true that the RAM capacity may be 256 Kbytes or 1 Mbyte depending on the configuration.

An "Elektronika MS 6105" black and white graphics display is used as the display; the video controller of this unit has the following operating modes:

- in the text mode: 25 lines of 80 characters each or 25 lines of 40 characters each;

- in the graphics mode: 640 × 200 and 320 × 200 pixels.

The capacity of the floppy disks used in the "Iskra 1030" computer is 360 Kbytes corresponding to the IBM PC standard. The "Winchester" hard disk has a 5 Mbyte capacity.

However compatibility with the IBM PC format is possible if the floppy disk is formatted for 40 tracks corresponding to 180 Kbytes. This format which makes it possible to write 360 Kbytes of information, while corresponding in size to the IBM PC standard, is in fact not compatible with this standard since the floppy disks in the "Iskra 1030" computer are formatted on one side while in the IBM PC they are formatted on both sides. Therefore the floppy disk format selected and implemented by the designers of the "Iskra 1030" machine provide the user with many "free" minutes during operation. This is particularly noticeable when transferring programs from the IBM PC or from the ES1840 or the "Neuron 66" computers.

We will examine this issue in somewhat greater detail. In order to read IBM PC-formatted disks (340 Kbytes) on the "Iskra 1030" they must be formatted by a format/1 command, which distributes 40 tracks on one side of the floppy disk. Then the information written on one two-sided magnetic disk must be rewritten on two single-sided 40-track diskettes on a computer utilizing the two-sided floppy disk memories.

The user can then choose to employ either single-sided 40-track magnetic disks or to rewrite information from these two disks onto one single-sided 80-track disk on the "Iskra 1030" which is done by means of a COPY command.

However there is one more surprise when the user finds out that since the ADOS operating system of the "Iskra 1030" is compatible with MSDOS it considers the single-sided 80-track magnetic disks to be two-sided 40-track disks. This is because the track carrying even numbers is read logically by the operating system as side 0 of a two sided diskette while the track with odd numbers is read as side 1.

It is interesting to note that the designers decided to mount the disk drives for the floppy disks vertically rather than horizontally. Users can decide whether or not this is convenient by using the computer.

It is true that another design is quite convenient: the decision by the designers to place on the front panel of the display the brightness and contrast controls, and especially the volume control which fans of computer games consider necessary. The power switch is located on the back panel of the system unit while, unfortunately, there are no operating system reset or reload switches.

The data transmission adapter supports interfacing with other computers on communications channels through connectors satisfying the radial serial interface standards, as well as a C2 connector. The data transmission adapter supports communications on telephone channels by means of a signal converter (a modem).

The "Iskra 1030" personal computer can support a coprocessor based on the K1810 VM87 microprocessor for performing floating point arithmetic operations.

The 8 Kbyte computer ROM consists of four 2 Kbyte integrated circuits. The ROM contains the initialization program for the operating system bootstrap routines. The ROM also contains the drivers (control programs) for the peripherals. The bootstrap routine also tests all devices in the personal computer.

The basic software for the "Iskra 1030" computer consists of:

- the operating system;

- the ADOS programming language;

- the MASM programming language;

- the BASIC programming language; and

- the YaMB (accounting machine language) programming language.

The ADOS operating system is an analog of the MSDOS operating system. The instruction mnemonics of the ADOS operating system are compatible with the MSDOS instruction mnemonic. The architecture of ADOS is analogous to that of MSDOS. The MASM macroassembler is designed primarily for system programmers that cannot imagine writing programs in any language other than ASSEMBLER. The YaMB programming language is designed for economic and statistical applications. It is used in the software simply to provide program compatibility and to support the programs that were written for the "Iskra 555" and the "Neva 501" accounting machines.

The SM 1800 personal computer family

The SM 1800 personal computer family includes the following models: The SM 1800, SM 1804, SM 1810, and SM 1814. The SM 1804 and SM 1814 computer models are dust resistant versions of the computer.

The SM 1800 personal computer is based on the 8-bit KR580 microprocessor set. The modularity principle underlies the design of the SM 1800 computer family, as with the computers examined above. The I41 universal system interface and the modularity of the architecture make it possible to integrate the modules and the system on the printed circuit board, which provides significant flexibility in the design of control systems and computer systems and also provides expandability of the computer as a system. The expandability property can be used for further expansion and development of the overall system.

A new component base - the KR 1810 microprocessor set - was used for the SM 1810 computer; this microprocessor set utilizes 16-bit data. The expandability and modular design of the architecture made it possible to use virtually all modules from the SM 1800 in the SM 1810 computer model

The SM 1800 computer architecture can support simultaneous operation of up to 8 central processor modules, each of which contains a microprocessor, a local RAM and ROM, and a connection to the I41 universal interface. Up to 16 such modules can be used in the SM 1810 computer.

The K 1810 VM86 microprocessor operates at a 4.77 MHz clock frequency and can address up to 1 Mbyte of RAM. A coprocessor for floating point arithmetic operations can be used in the SM 1810 computer.

The microprocessor system based on the K 1810 microprocessor set and the I41 universal interface has the following advantages over a single process system:

- a system task is divided into a number of subtasks, each of which is executed by its own dedicated microprocessor;

- several microprocessors can operate simultaneously, thereby increasing the operating speed of the overall system.

The address bus in the SM 1810 computer has been expanded from 20 bits (SM 1800) to 24 bits which makes it possible to address up to 16 Mbytes of RAM. The majority of modules from the SM 1800 can be used in this personal computer. The SM 1810 computer contains an auxiliary 8-bit module to establish compatibility between these modules.

The SM 7209 and VTA22000 terminals are used as displays; these support operation in an alphanumeric mode producing 25 × 80 characters.

It should be noted that there is a broad variety of software in the SM 1800 computer family. This software includes a general purpose operating system, a resource (training) system, real-time operating systems, and the BASIC software package.

The OS 1800 general purpose and micro-DOS operating systems contain a rather broad set of programming packages. These packages are compatible with the CP/M operating system. They consist of the basic disk operating system (BDOS), the basic input-output system (B I/Os) and the console command processor.

The resource software is designed to develop complex programs, largely for real-time systems. These include the DOS 1800 and DOS 1810 systems. The programming languages here are BASIC, FORTRAN, and PASCAL. The BASIC real-time system supports multiprogram operation.

The Mikros-86 operating system which is compatible with the CP/M-86 operating system, is used.

The user has access to a wide range of program packages. The "Dialog" package makes it possible to develop and utilize an information retrieval system and to produce data bases. The "data preparation" package supports data processing in documents in any format. The BnD (data base) package is used to produce relational data bases. The "Kalendar" package is used to supervise management functions.

The YeS series personal computers

It is likely that everyone is familiar with the mainframe YeS computers. This family has also been expanded to include new personal computers. The YeS 1840 professional computer provided access to the rich existing library of programs.

Structurally the YeS 1840 computer system consists of several functional modules. Above all this includes the system electronic unit, the floppy disk set, the display (a black and white graphics display in the most recent models) and a keyboard.

The system unit is 150 × 300 × 450 mm in size. It contains the basic system board, the RAM board, the display address board, the floppy disk drive adapter board, and the interface adapter board. The base system unit also contains the power supply and a fan. Additional peripheral adapter boards or an additional RAM expander board can be connected to the free connectors on the system bus (acquiring such boards is rather problematical).

The expander unit is included in order to support a large number of peripherals. The expander unit is based on the same structural design as the system unit. It has its own power supply and, like the system unit, has a system bus and seven connectors. It is true that there are only six free connectors since one is assigned for connection to the system unit.

The "heart" of the system board is the K 1810 VM86 microprocessor which can handle 8- and 16-bit data (an analog of the Intel 8088 microprocessor) and can address a memory of up to 1 Mbyte by means of a 20-bit address. However the RAM capacity available in the basic configuration is only 512 Kbytes (you will soon find out whether or not this is enough when using the computer). The microprocessor contains a 6 byte buffer to increase information handling speed, which makes it possible to select the next commands while the microprocessor is handling the present command.

The computer ROM has an 8 Kbyte capacity. Computer test programs are contained here; these are executed when the computer is turned on. The disk drives used in the YeS 1840 are designed for 130 mm floppy disks and write in the 360 Kbyte format, i.e., 2 × 360 Kbytes. This is, of course, small and therefore a program is available to expand the information capacity of each floppy disk to 720 Kbytes. This is possible as each magnetic disk is considered to be two disks, which maintains compatibility with the IBM PC standard.

In the early YeS 1840 models an alphanumeric display was used to produce 25 lines of 80 characters each. However it was subsequently replaced by a black and white graphics display (the same unit used in the DVK-3 and in the "Elektronika MS 0585"). The operating modes make it possible to obtain up to 16 gray shades and a maximum resolution of 640 × 200 pixels.

The YeS 1840 keyboard employs a planar design structure containing 92 keys with a key configuration similar to that of the IBM PC.

The floppy disks are housed in a separate unit which of course complicates the layout of the computer on a desk. The unit contains two stacked memories. An independent power supply is used so that in switching on the computer the wisdom of the designers is often recognized since the power supply switches are located on the rear panel of both the system unit and the memories. No "hot start" key exists either. In most cases it is possible to reload the operating system by simultaneously keying the CON/ACC/REM keys [UPK/DOP/UDL] although it is possible to determine soon enough whether or not this is insufficient.

One additional inconvenience is the design of the cables and connectors used in the computer. A screwdriver is required to connect different units while interconnecting cables are required to cover the substantial space for containing such units.

Auxiliary modules have been developed to expand the capabilities of the YeS 1841.42 computer: a discrete signal input-output module, an analog signal input-output module, a CAMAC system interface module, and a digital speech synthesizer.

An "Estafeta" network adapter and a YeS 7920 system adapter module have been developed for establishing local area networks. The C2 connection adapter (an analog of the RS232 permits connection to a remote,data processing network based on modules of the mainframe YeS computers through multiplexers and modems. The YeS 7920 system adapter will support connection of the YeS family personal computer to the 7920 system as one of the terminals of this system, thereby providing access to the resources of a mainframe YeS computer. The adapter of the "Estafeta" local area network can be used to set up ring-type local area regional computer networks. The "Estafeta" network can interconnect up to 125 personal computers with a maximum distance of up to 125 km between computers.

So you have already installed a YeS 1840 personal computer (and possibly an 1842) at your workstation, have switched on the power and have started working. What basic operating system are you to use? The manufacturer has provided with the standard YeS computer set the M86 operating system and a set of applied program packages. The commands in the M86 operating system have Russian notation and the program packages will "speak" with you in Russian. It is probably best if you carefully review program documentation at a convenient site with the diskettes on which the software is written. You can then acquire the MSDOS from your friends or through official channels. You ask why?

Such a favorable operating system as the M86 is a close brother to the CP/M-80 operating system. However, this is in the past.

Program compatibility with the IBM PC and its MSDOS operating system in fact will make the YeS 1840 such a powerful tool that it becomes indispensable. The MSDOS will allow the user access to all such software currently available. You can use such modern program interfaces as PCTOOLS and NORTON COMMANDER. You will have access to a wide variety of modern programming languages from BASIC through PROLOG, data base management systems and integrated tools packages. So do not waste your time on M86 and begin immediately with MSDOS.

One additional comment. The capabilities provided by modern operating systems such as MSDOS and their software require substantial RAM capacity and a hard disk. Therefore you will soon find out that, unfortunately, the YeS 1840 computer does not permit utilization of all tools provided by the software.

THE VECTOR PROCE'SOR OF THE ELBRUS-2 MULTIPROCESSOR COMPUTING COMPLEX

907G0089A Moscow SUPER-EVM in Russian, pp. 1-29

[Article by V. S. Burtsev, Ye. A. Krivosheyev, V. D. Asriyeli, P. V. Borisov and K. Ya. Tregubov]

[Text] 1. Principles of Design of the Vector Processor and Its Operation Within the Elbrus-2 MCC

Development of the Elbrus 2 MCC was mainly oriented at achieving utmost performance of the complex in solving complicated scientific data processing and computing tasks in a collective use modality and real time problems. The work was carried out in the following directions:

- achieving utmost performance of the universal central processors by having a parallel (in time) computing process by dynamic allocation of resources such as actuating devices, the fast registers of the scratch-pad memory, and so forth;

- use of a modular multiprocessor architecture with dynamic allocation, especially allocation of the modules of central processors, memory, I/O processors, and exchange channels with the external memory and the external device modules;

- widespread use of specialized processors in the multiprocessor system, including first and foremost a vector processor.

In order to understand the features of using specialized vector processors alongside the universal central processors in a multiprocessor architecture, let us consider the basic principles of their functioning, especially as it enables a faster computing process.

It must be mentioned that a specialized vector processor is capable of hastening a computing process that represents a parallel graph of non-interconnected homogeneous operations on data. Any given operation on data (addition, multiplication, divisior etc.) can be broken up into an interrelated sequence of smaller operations, such as subtraction of exponents, matching of exponents, taking the reciprocal, obtaining the half sum and carry, and other microoperations. If the operations are not interrelated in data, e.g., if it is required to add vectors component by component, before the operations on the first paid of vector components are completed and after finishing a particular microoperation it is possible to begin the addition of the next pair of vector components, and so forth. If the operation of addition is broken up into N_{mop} consecutive microoperations, the times of execution of which (τ_{mop}) are equal to each other, then after a time $N_{mop} \times \tau_{mop}$ all the devices performing the microoperations will be engaged in working on the vector components. In this case, the speed of the adding device will be N_{mop} faster than that when working with single interrelated data.

The effect of the possible increase in speed when working with vectorized data may be augmented by considering simultaneous execution of not one, but several different (including data-interrelated) operations on vectors. Thus, for example, if it is needed to perform the following sequence of operations on vectors:

$$R := \frac{A \times B + C \times D}{E},[1]$$

we may organize a vector processing "pipeline," as shown in Fig. 1. In this case, all the devices performing microoperations will be full after a time $20\ \tau_{mop}$. It is assumed that the multiplication device performs its operation in six, the addition device in four, and the division device in ten τ_{mop}, and all of these have the appropriate number of consecutively working microoperation blocks. The thus-organized vector processor in the above example will be 26 times faster than an ordinary scalar processor, consecutively handling the operations by the identical formula:

$$r := \frac{a \times b + c \times d}{e}[2]$$

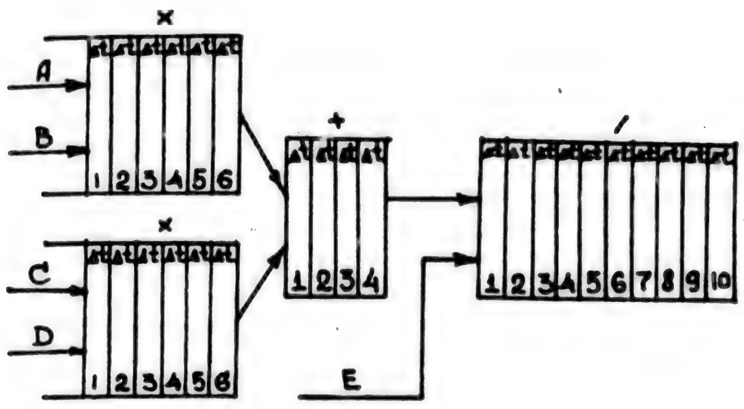

Fig. 1. A vector processing pipeline.

Even if the scalar processor has two multiplication devices and the multiplication operations are performed at the same time, its speed will still be

[1] R, A, B, C, D, E - vectors having n components. Here and below, component operations with vectors are understood.

[2] r, a, b, c, d, e - scalars.

20 times less than that of the vector processor. However, to achieve such performance, the vector processor must have a strictly (accurate down to τ) synchronized input of operands and the devices for performing the computations must be free at the required time. Satisfaction of these demands is achieved by total monopolization of the resources of the processor and working storage on behalf of solving the single problem. Thus, simultaneous operation of several processors with a shared working storage and multiprogram operation with dynamic distribution of the resources in the course of solving a problem are excluded. In such conditions, when using a wide-format instruction that actually consists of several operations, it is possible to perform a (static) distribution of the resources over time during the process of translation with accuracy down to an interval τ_{mop} in order to evaluate different vector expressions, hereafter known as macrobased crosses.

Inevitable equipment standstills occur when switching from the running of one macrobased cross to another. Therefore, the longer the vectors, the fewer such losses. Thus, for a vector 20 elements long, the efficiency in using the equipment for the above example cannot be more than 50 percent, and consequently speed of the processor is reduced by half. The losses in speed are associated with equipment standstill during the time of filling of the pipeline and its emptying, in other words, during the "runup" and "slowdown." To be sure, these times are functions of the time to read the vector from memory--the memory access time (T_{acc}). Thus, in the CRAY-1, eight directly-addressable fast vector registers (64 elements each) have been introduced between the memory and the processor in order to shorten the time of runup and slowdown, which has greatly increased the speed of the complex for short vectors, as compared to the computer Cyber 205 [13, 14, 15, 16]. Yet introducing such small number of directly-addressable registers has limited the capacity of the CRAY-1 to create different pipeline configurations adapted to the macrobased crosses and made the programming much more complicated. Therefore, it is better for the processor to work directly with local RAM, provided that the times of runup and slowdown are substantially reduced.

If the system is strictly determined in time, it is possible to delegate to the translator the task of allocation of hardware resources during the performance of the macrobased cross, as well as the task of preparing the next configuration of the processor while the old one is still running. This involves a preliminary fetching of operands from the local memory and a using of the actuators as they become available--the stage of slowdown is integrated in time with the stage of startup of the new macrobased cross.

It should be noted that the effectiveness of this method depends largely on the quality of the translator, since the processor-memory resources are allocated during the translation and before the program is executed. The task of the translator is substantially complicated when the vectors are small in dimension, $n < N_{mop}$, where n is the number of elements of the vector to be executed by the macrobased cross, and N_{mop} is the number of microoperations of the pipeline adapted for this cross. When $n = 1$, the efficiency of this method is greatly reduced, since the pipeline must be reconfigured in each clock period for sufficiently complete utilization of the pipeline.

It is necessary to observe yet another feature of the operation of a vector processor. Suppose that the pipeline configuration shown in Fig. 1 is operating. Then, in each clock period (τ_{mop}), five numbers should be read from memory and one written. If $\tau_{mop} = 12.5$ ns (as in the CRAY-1), the memory retrieval cycle should be ≈ 2 ns.

To solve rather large problems, a memory of tens of millions of words is required. It is virtually impossible to construct a memory of the required volume, operating at a speed of several nanoseconds, without substantial forfeiture in the memory access time (T_{acc}). The question therefore arises as to the volume of the scratch-pad memory of the vector processor, which should be such that the rate of exchange with the next memory level is at least an order of magnitude smaller thanks to repeated use of the data located therein.

Taking a scalar processor as an analogy, this condition is met with a scratch-pad memory or cache memory volume of several thousand words. Accordingly, for a vector processor with average vector length of 100 elements, we may assume a scratch-pad memory volume equaling several hundreds of thousands of words. Thus, a vector processor must have a <u>local</u> scratch-pad memory much larger than the memory of a scalar processor. Of course, such different methods of organization of the computing processes for scalar and vector arithmetic requires fundamentally different architecture and circuitry in the design of the computer hardware, specifically:

- the scalar processor should have a minimum time of execution of operations and time of memory access. The vector processor should have, first and foremost, a maximum speed of computations and maximum memory throughput capacity [1];

- the hardware of the scalar processor should ensure effective operation in multiprogram mode. The vector process should be oriented to solving large problems in monoprogram mode;

- the most efficient method of allocation of resources (actuators, registers, etc.) of a scalar processor, as shown by the experience with the use of the Elbrus MCC, is dynamic (as the programs are being executed), to be accomplished (generally) by hardware. Effective execution of programs in the vector processor requires a static allocation of resources during the translation of the programs, which should be secured by completely different hardware strategies in the design of the processor.

The integration of such conflicting demands in a single processor leads to relatively incompatible circuitry structures of the processor design. The inevitable compromise solutions must reduce the maximum speed of either the vector or the scalar portion of the processor. Therefore, the solution to the problem of achieving maximum speed is best achieved in a multiprocessor complex with extensive use of specialized vector processors, alongside universal processors (oriented to scalar computations). This technique of vectorization was selected in the creation of the Elbrus-2 MCC.

A similarly designed complex is the supercomputer Cyber 205, which in its full array has four vector processors and one universal processor. The Elbrus-2 with vector processors would have certain advantages over the Cyber 205, thanks to the possible creation of an array consisting of any desired combination of vector and scalar processors and the presence of a very high speed local memory in each vector processor. In this way, the speed of the Elbrus-2 MCC with vector processor would be comparable to that of the

Cyber 205 in many applications, despite the fact that the componentry of the latter is nearly three times faster.

The architectural features of a vector processor are important when it is working in an array. Thus, while the central processors in the Elbrus-2 MCC are loaded during the execution of the computation by hardware implementation of the principle of depersonalized operation, the loading of the vector processor should be done by the complex with program control. In order to secure a strict synchronization of the data arriving at the inputs of the actuators, and also substantially reduce the time of working with the common memory, the vector processor should be coupled to the complex via a local memory of sufficiently large size.

A block diagram of the vector processor in the Elbrus-2 MCC is shown in Fig. 2.

In designing the vector processor, it was extremely important to determine the necessary memory volumes Q_1 and Q_2, in order to satisfy the relations:

$$K_1 = E_1/E_2 \text{ and } K_2 = E_2/E_3,$$

which obtain in the solution of the majority of problems.

The maximum handling capacity of the vector processor E_1 = 1 word/5 ns can be determined from the operation of the VP under maximum workload of its actuators (Fig. 3). The value of E_2 should not exceed the maximum rate of exchange of the central processor (E_2 = 1 word/80 ns) [3]. Thus, K_1 should be greater than 16.

If we grant that five disks can work at the same time with a single vector processor, then E_3 will be determined by the total average exchange rate of 2 μs per word and the value of K_2 should be no more than 25. The possibility of fulfilling these relationships has been checked by a model investigation of the Navier-Stokes problem (see below and [7]), which also determined the volumes of the local memory, Q_1 = 500,000 words, and the working memory Q_2 = 16 million words.

The delegation of vector arithmetic to a separate processor also has negative aspects, to be sure. One of these is that purely vector or purely scalar computations do not commonly occur, and thus there may be losses in transfer of data from the vector to the scalar processor and vice versa. These losses are moderated by the following factors:

a) the central processor of Elbrus-2 has hardware support for the execution of vector computations, thus increasing the speed in the vector processing cycles by 10-20 percent, and the vector processor is able to perform scalar computations using the very same actuating devices;

b) the time to transfer data from the vector to the scalar processor and back has been reduced to the utmost: the transfer of 10,000 words from the vector to the central processor and back occupies no more than 1 mls (the maximum handling capacity for the entire working memory is 5 ns [3]);

c) successful projects are being accomplished to increase the degree of vectorization of algorithms while at the same time localizing them. In this case, the scalar portion of the computations can be done in the background of the vector portion [10, 11].

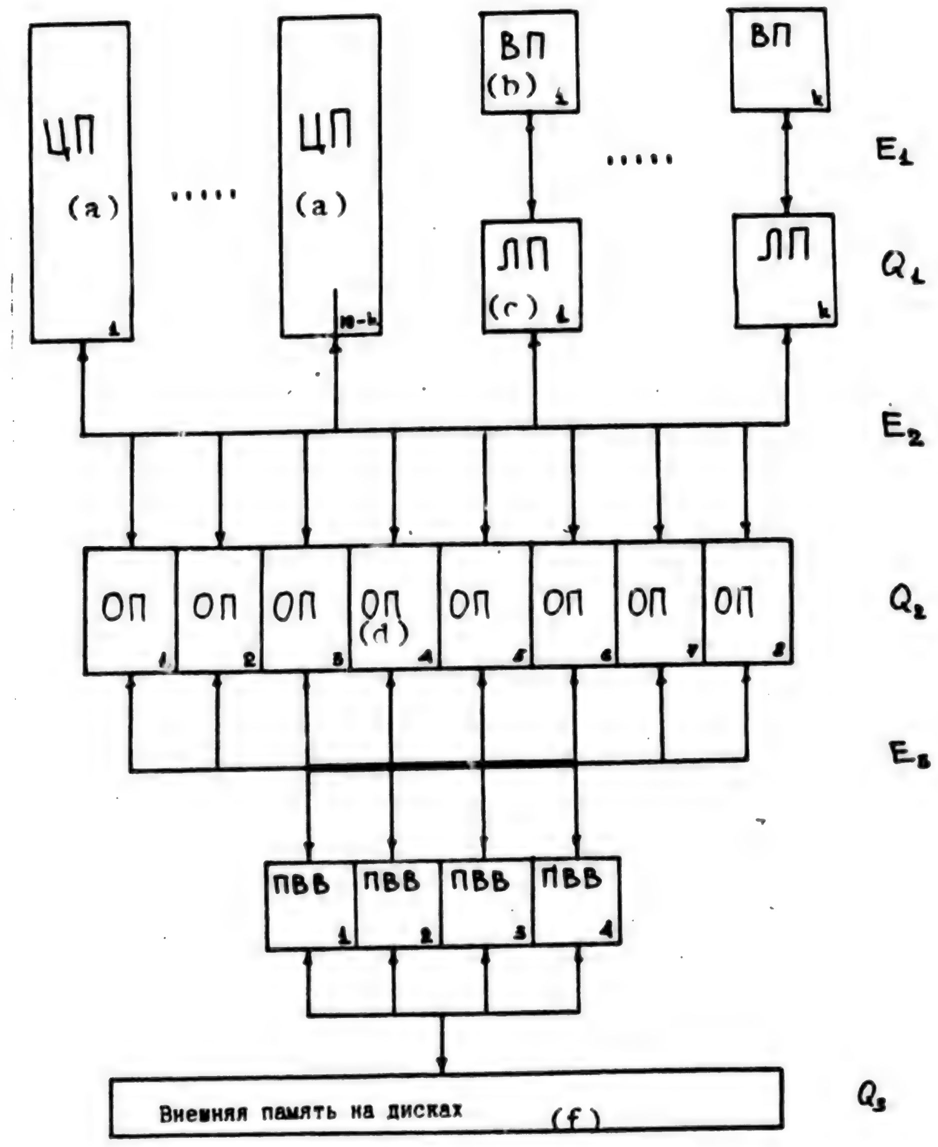

Fig. 2. Block diagram of the Elbrus-2 MCC with vector processor.

Key:

a.	Central processor	d.	Working memory
b.	Vector processor	e.	External disk memory
c.	Local memory of vector processor		

Note: E_1, E_2, E_3 - average handling capacity of communication channels; Q_1, Q_2, Q_3 - volumes of local, working, and external memory.

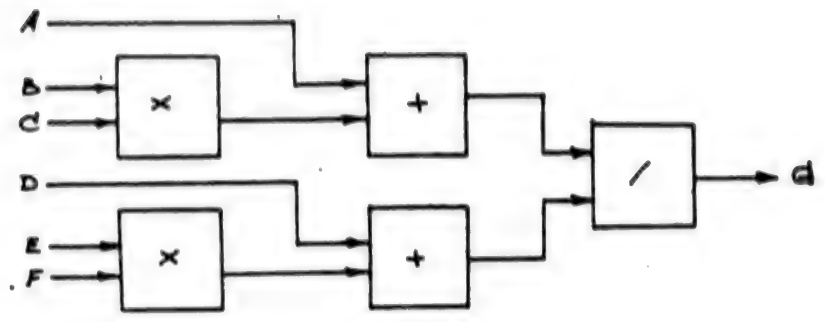

Fig. 3. Configuration of a vector processor pipeline making maximum use of the dynamic resource of working memory [1].

New and original architecture solutions have been proposed in the development of the vector processor, substantially improving the effectiveness of utilization of its actuating devices. These include, first and foremost, firmware principles of reconfiguration of the processor in order to adapt it to execution of different macrobased crosses, juxtaposition in time of the preparation of the next processor configuration during the completion of running of the previous configuration, or time integration of the stages of startup and slowdown, the handling of the "read vector" instruction immediately after the "write" instruction, and so on.

What is new and totally original is the high level language, which along with a natural description of the algorithms ensures effective utilization of the equipment of the vector processor. New methods have been developed for the organization of the optimizing translator. The project was carried out in view of the experience from debugging and operation of mainframe computer systems [5, 6].

All of these new solutions will surely be used extensively in future Soviet supercomputer projects. A detailed description of the most interesting and original architectural, circuitry and software solutions is presented in the current work and in the respective articles of the collection.

2. Architectural Features of the Vector Processor

Two vector processors--the CRAY-1 and Cyber 205--are currently in greatest use. It is therefore convenient to relate the design features of the vector processor to these architectural solutions [13, 14, 15, 16].

As already pointed out, one of the methods of raising the speed of a vector processor is the organization of a parallel operation of its functional arithmetic executing devices by connecting them into a processing pipeline so that the outputs of certain devices are connected directly to the outputs of other devices (see Fig. 1). Hereafter, we shall call this concatenation.

The CRAY-1 has three functional vector devices that can be concatenated. The concatenation is done in hardware fashion (the programmer may not even be aware of it), and therefore the intermediate results (despite the concatenation) should be entered into the vector registers, which lessens the overall speed of the vector processor.

In the Cyber 205, high performance is achieved by parallel operation of several processing pipelines (as many as four). In the processing pipeline itself, two devices may be concatenated, for which purpose an explicit statement is required in the program, the LINK instruction in front of the two operations being concatenated.

In our vector processor (VP), as many as six functional devices can operate simultaneously:

- two addition devices;

- two multiplication devices;

- a division device;

- a logic device.

In the vector processor, the programming of a particular formula is done not by a sequence of operations, but by specifying the connections among the group of functional devices performing these operations. The inputs of each of the six devices may be connected directly to the output of any given device. In order to compute, for example, the vector expression:

$$G := \frac{A + B \times C}{D + E \times F} \qquad (1)$$

where A, B, C, D, E, F, G are vectors having n components, we may construct a processing pipeline with the configuration shown in Fig. 3. The overall speed of the VP in this case is $5/\tau$, where τ is the work pace of each functional device.

The vector processor, like the Cyber 205, works directly "from memory into memory" (the operand is read from main memory and the result is written into main memory), which substantially enlarges its capabilities and simplifies the programming as compared to the CRAY-1. However, unlike the Cyber 205, each processor of the VP in the Elbrus-2 has its own local memory--the vector processor memory--that is connected by standard processor-memory channel to the common main memory of the latter (see Fig. 2). As a result, the VP has six read-from-memory flows and two write-in-memory flows. This is sufficient to construct almost any reasonable configuration from the available group of functional devices.

The two write flows make it possible to construct two parallel-operating processing pipelines (Fig. 4) to compute, e.g., two expressions:

$$D := A \times B + C\ ; \qquad H := E \times F + C.$$

The handling capacity of the vector processor memory, which should provide for the overall flow of the processing pipeline, is achieved by an interleaving of the memory at the low-order bits of the address for the parallel-operating modules.

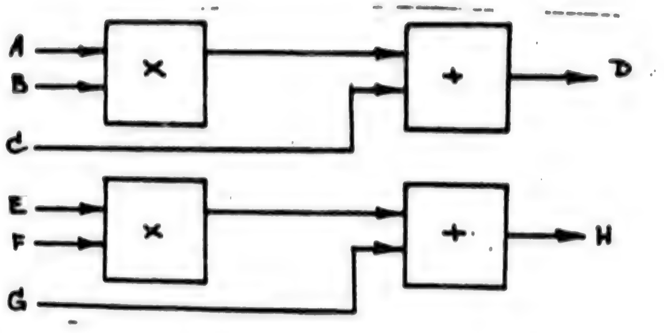

Fig. 4. Two parallel-operating pipelines.

So that no limits are placed on the mutual arrangement of the vectors used by the processing pipeline in the memory, the flows are serviced in alternate batches of k numbers, where k is the number of memory modules. But this is only possible for a particular spacing at which the vector components are lodged in memory.

In the Cyber 205, simultaneous accessing of memory modules by independent addresses is not possible, and therefore the processing pipeline is linked to the memory by superwords with a spacing equal to unity. This is a major limitation that reduces the effectiveness of working with multidimensional arrays.

In the VP memory, the modules are accessed by independent addresses, thus substantially moderating the demands on the spacing of the vector (with respect to difference in the addresses of consecutive vector components). If the number of memory modules equals a power of two, the vector spacing should be an odd number. This guarantees that any consecutive k components of the vector will be arranged in different memory modules. Such organization of the interaction with the main memory enables an effective working with multidimensional arrays of data for all measurements. For example, if it is necessary to work with the rows and columns of a matrix, it may be necessary to introduce an imaginary column to make the row spacing and the column spacing odd numbers.

The VP has a private buffer for each read and write flow. The buffer of the input flow from memory receives k numbers at the same time, and one number is sent out to the input of a particular functional device in each clock period. On the other hand, the buffer of the outgoing flow accumulates one word from each output of a functional device and sends out k numbers for writing into memory.

One of the major deficiencies of Cyber 205 is the poor effectiveness of working with short vectors. Thus, when the length of the vectors n is equal to 25, Cyber 205 puts forth 20 percent of its maximum performance; if the length is 100, it puts forth 50 and if the length is 1000, it puts forth around 90 percent. Such asymptotic relationship is the result of the great inertia of the pipeline and the losses during startup and slowdown.

In the vector processor, special attention has been given to minimizing the performance losses associated with clearing the pipeline devices upon completion of work in one configuration (slowdown phase) and filling the pipeline of the other configuration (startup phase) by introduction of firmware to integrate the startup and slowdown phases, thus increasing the efficiency of the VP when working with short vectors.

Let us consider the functional layout of the vector processor (Fig. 5) in order to explain the implementation of the new principles of processing of vector information. The processing pipeline includes:

- six functional arithmetic pipeline devices;

- six incoming flow buffers;

- two outgoing flow buffers;

- four real register files designed to hold constance and scalar variables.

When the processing pipeline is assembled, the links between the above devices are established by the configurator, which makes it possible to connect each input of the functional devices, the register files, or the outgoing flow buffers to the output of any given functional device, register file, or incoming flow buffer.

The configurator is controlled by the configuration control unit in accordance with the flow of configuration-change instructions arriving from the instruction buffer. The other instructions are sent from this buffer to the index processor, which is a universal integer processor designed to organize the computational process and the preliminary processing of the address instructions and exchange instructions. This has a 32-bit arithmetic-logic unit and its own internal memory of 1K word capacity, which may be transcribed into the VP memory through the write buffer and unloaded from the VP memory through the read buffer. In addition, the index processor has access to the real register files.

The exchange device organizes an exchange between the VP memory via the input and output buffers and the main memory of the MCC, to which the exchange device is connected across a standard interface in place of one of the central processors of the MCC.

The VP memory with a cycle of 40 ns has an interleaving of eight independently addressable modules. Requests for the VP memory are handled in keeping with their priority. The highest priority belongs to the write flows, after which come the instruction flow, the read flows, the swap flows of the index processor memory, and the exchange flow.

The address unit is designed to generate the physical addresses for access to the memory modules of the VP from all users, employing the current starting address and the spacing of the flow being handled at the given moment to generate the eight addresses.

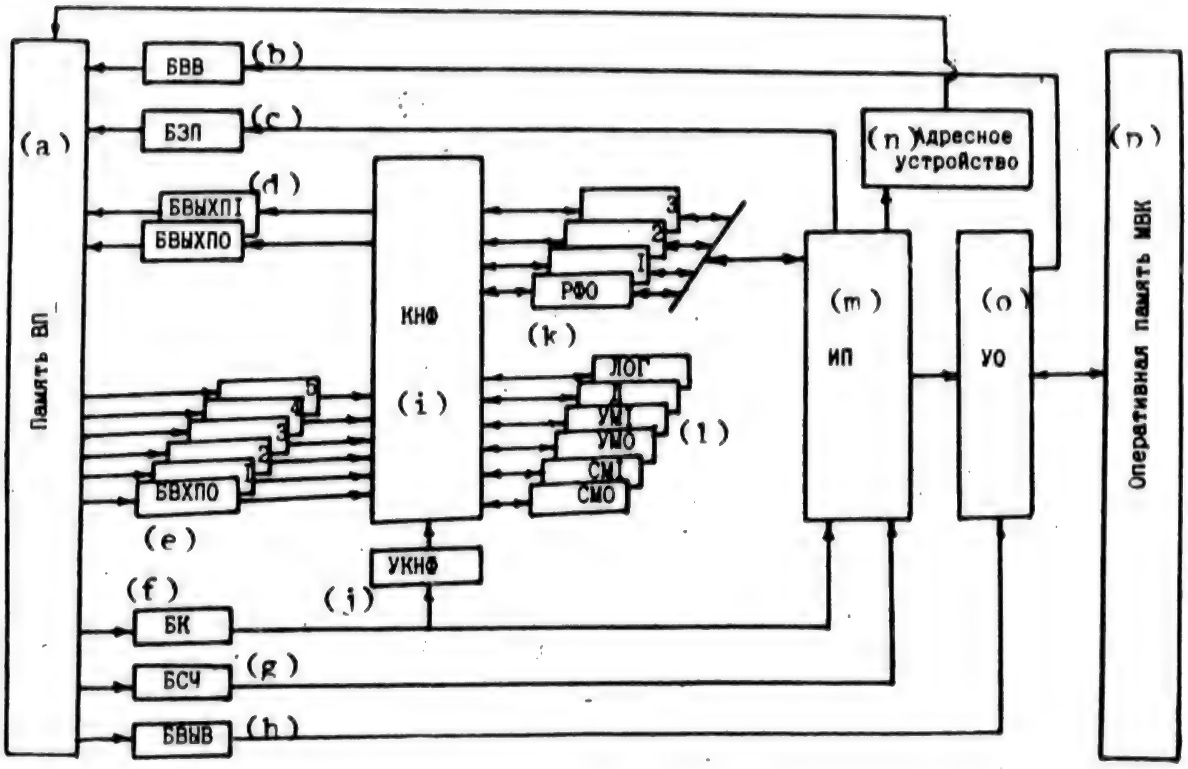

Fig. 5. Functional diagram of the vector processor.

Key:

a.	VP memory	k.	Real register files
b.	Input buffer	l.	Functional devices (logic circuit, divider, multiplier 1, multiplier 0, adder 1, adder 0)
c.	Write buffer	m.	Index processor
d.	Outgoing flow buffers	n.	Address unit
e.	Incoming flow buffers	o.	Exchange unit
f.	Instruction buffer	p.	Main memory of MCC
g.	Read buffer		
h.	Output buffer		
i.	Configurator		
j.	Configurator control unit		

The vector processor uses the following representation of numbers, address system, and instruction system.

1. REPRESENTATION OF NUMBERS. There are two formats for representation of numbers in the VP instruction system:

- 64-bit format of numbers with floating decimal;

- 32-bit format of integers.

All functional pipeline devices work only with 64-bit numbers.

2. ADDRESS SYSTEM. The vector processor has two kinds of memory: main vector memory and register files. Both the index processor and the processing pipeline may access the register files.

The information in the vector memory is arranged in arrays. Programming of all access to the vector memory is done in mathematical addresses. The physical addresses of the instructions and data are generated by adding the mathematical address to the value of the particular base register.

3. INSTRUCTION TYPES. a) The instructions of the index processor include:

- integer arithmetic operation;

- logic operation;

- conditional and unconditional jump instructions;

- transfer instructions addressed to the register files;

- the instruction for loading the vector length counter;

- special instructions for hardware support of the programming system.

b) Address instructions enable addressing of scalars, one-dimensional and two-dimensional arrays. When addressing a one-dimensional array, the initial address, interval and quantity are specified; when addressing a two-dimensional array, the initial address, interval for the first measurement, quantity for the first measurement, interval for the second measurement and quantity for the second measurement are specified.

c) The exchange instruction, like the address instruction, specifies an exchange array in the vector memory and, accordingly, an exchange array in the RAM of the MCC, as well as the exchange direction (input-output).

d) Reconfiguration instructions describe the process of reconfiguration of the processing pipeline over time.

As an example, let us consider the program for evaluation of the expression:

$$D := (A + B) \times C,$$

where A, B, C, D are vectors of length n. The program consists of three parts.

The first part contains the address instructions for computing the sequence of addresses of the components of each vector and loading the vector length counter.

The second and third parts consist of reconfiguration instructions: the second part describes the startup phase--the consecutive process of building the configuration for evaluation of the given expression, the third describes the slowdown phase--the consecutive process of clearing the devices upon completion of computation of the expression. Hereafter, such program for evaluation of a vector expression containing the aforesaid three parts will be termed a macrobased cross. The processing pipeline for evaluation of the macrobased cross with the specified formula is shown in Fig. 6.

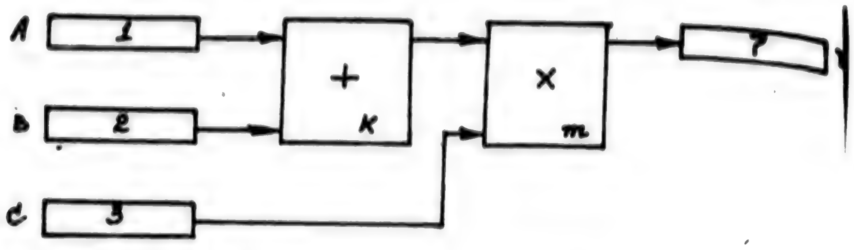

Fig. 6. Pipeline for evaluation of a macrobased cross.

Say the length of the pipeline of the addition device is equal to k clock periods, while the length of the pipeline of the multiplication device is equal to m clock periods. The startup phase of the macrobased cross may be described as follows.

1. Connect inputs of the adder to the incoming flow buffers 1 and 2, enable flows 1 and 2.

2. Delay for k clock periods.

3. Connect first input of multiplier to output of adder, second input of multiplier to incoming flow buffer 3, enable flow 3.

4. Delay for m clock periods.

5. Connect output of multiplier to outgoing flow buffer 7, enable flow 7.

After n clock periods from the onset of the startup phase, the slowdown phase of the macrobased cross should be initiated, which can be described as follows.

1. Disable the exit of incoming flows 1 and 2 from the buffers, clear the input of the adder.

2. Delay for k clock periods.

3. Disable exit of incoming flow 3 from buffer, clear input of multiplier.

4. Delay for m clock periods.

5. Clear input of outgoing flow buffer 7.

The reconfiguration instructions include:

a) instructions for partial reconfiguration of the startup phase, which describe the dynamics of change in configuration of the startup phase over time. The instruction indicates the connections that should be established between the devices and how many clock periods the configuration should remain unchanged;

b) the instruction for the time diagram of the startup phase of the macrobased cross, which indicates (for each device involved in the processing pipeline) the delay in clock periods from the onset of the startup phase till the time when the device is required in the configuration;

c) the instruction of the time diagram of the slowdown phase of the macrobased cross, which indicates (for each device involved in the processing pipeline) the delay in clock periods from the onset of the slowdown phase till the clearing of the input of each device.

The partial reconfiguration instruction is local in action, i.e., it changes only those connections in the processing pipeline that are explicitly stated (everywhere else the configuration remains as before). The quality of localness makes it possible to combine independent startup and slowdown phases. The startup phase of a macrobased cross allows a gradual building of the configuration--from the incoming flow buffers to the required functional devices at the exact time when they come into actual use. This makes it possible to begin the building of the next configuration without waiting till the end of the computation in the previous configuration, and to make use of the devices in the next clock period after the clearing of their input by the previous configuration.

The instructions of the time diagrams of the startup and slowdown phases have been specially introduced to achieve a hardware integration of two contiguous macrobased crosses. The slowdown phase of the cross begins in n clock periods after the onset of the startup phase, where n is the length of the vector.

It is natural to require that the reconfiguration program of the macrobased cross be independent of the length of the vector. However, for short vectors, it is possible to combine the startup phase and the slowdown phase of the cross. This situation is diagrammed in Fig. 7a. In the interval Δ, it is necessary to execute two flows of reconfiguration instructions at the same time.

In order to integrate in time the slowdown phase of the current macrobased cross with the startup phase of the following macrobased cross, it is no longer sufficient to execute only two flows of reconfiguration instructions at the same time, since the situation shown in Fig. 7b is possible in short vectors. In the interval Δ, three flows of reconfiguration instructions should be executed at the same time.

The vector processor realizes three simultaneous flows of reconfiguration instructions. The starting of the next macrobased cross may begin after completion of the startup phase of the current cross, i.e., the length of vectors in which it is possible to work without loss is equal to the length of the processing pipeline (on the order of 20-30).

As for any other pipeline working "from memory into memory," the problem of read after write (RAW) arises in the VP, the nature of which is as follows. Say a macrobased cross is using the result of a previous cross. For proper functioning, the execution of the second macrobased cross must not begin until the pipeline is emptied and all results of the first cross are written into memory, i.e., it is not permitted to combine the startup and slowdown of two such macrobased crosses.

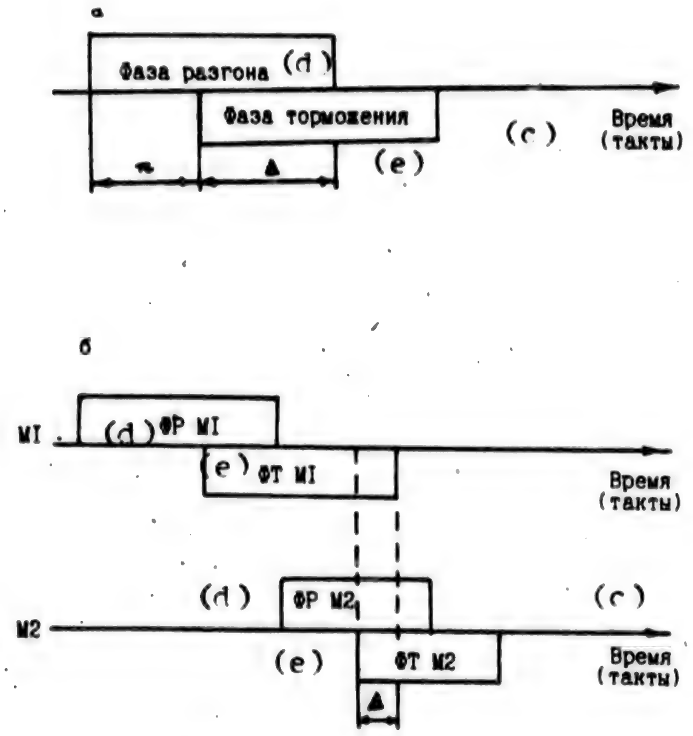

Fig. 7. Time diagrams of operation of a vector processor in the startup and slowdown stage: a - one macrobased cross; b - two consecutive macrobased crosses.

Key:

c.	Time (clock periods)	d.	Startup phase
		e.	Slowdown phase

In the VP, the RAW problem is solved in firmware fashion. For each array, the address instruction contains information about the range of variation of addresses (R). The programming system guarantees that the addresses of all elements of the array are contained in the segment [AB, AB + R], where AB is the address of the beginning of the array. If a segment of addresses of the read array of the next macrobased cross intersects a segment of addresses of the write array of a previous cross, the reading of elements of the particular cross is postponed in hardware manner until all results of the previous cross are written into memory. In other situations, combination of crosses is permitted. Comparison of the segments is done by the hardware, prior to fetching the operands in memory.

The architectural solutions adopted ensure maximum speed of the processor in the componentry of the Elbrus-2 MCC (τ = 40 ns) equaling 125 megaflops per second.[3] The actual speed of the processor was evaluated in a model investigation of the solving of the Navier-Stokes equations in a three-dimensional domain. This same problem was also used to clarify the required volumes of vector and working memory (Q_1 and Q_2) and the required handling capacity of the communication channels (E_1, E_2, E_3) (see Fig. 2).

3. Results of Model Investigation of the Solving of the Navier-Stokes Equations in a Three-Dimensional Domain in the VP

The choice of the problem of solving the Navier-Stokes equations in a three-dimensional domain as a test case for the vector processor was dictated by the following main factors:

First, the problem of solving three-dimensional Navier-Stokes equations with an increased number of nodes, beyond the power of the available computers, does exist and is urgent. Second, algorithms have been developed today for parallel solving of three-dimensional Navier-Stokes equations that are suitable for implementation in a vector processor [8, 9]. Third, the Navier-Stokes problem bears features of a typical problem with regulator grid, which places methodological value on this programming test.

The programming of the Navier-Stokes problem was done in the underlying programming language and pursued the following goals.

1) to assess the possibility of pipeline parallel processing of the problem;

2) to assess the volume of the vector memory needed for effective implementation of the algorithm;

3) to estimate the time to solve the problem;

4) to evaluate the effectiveness of utilization of the processor equipment in the solving of the problem.

It should be pointed out that solving such problem on a grid of 10 x 10 x 15 nodes in the BESM-6 takes about one hour, and increasing the number of nodes is not permitted by the low capacity of the working storage. Solving the problem in the BESM-6 on a grid of 50 x 50 x 50 nodes is almost impossible, yet it is just such problems that are becoming very urgent. We now examine this problem. The solution algorithm is based on the method of partitioning of spatial variables and physical processes and employs an algorithm developed at the ITPM SO AN SSSR [8, 9].

In the algorithm, by a special transformation of coordinates, the physical domain with irregular grid is turned into a cube with side N with uniform grid, in which the calculation is performed. Each time step of the solution is broken up into six consecutive minor steps. Six of the minor steps include

[3] $\tau = 40$ ns is the nominal clock period of the vector processor. The operating clock period of the Elbrus-2 MCC is $\tau = 44$ ns. In the model investigation of the working of the vector processor with Navier-Stokes equations, it was assumed that $\tau = 44$ ns.

two steps apiece in each of the three orthogonal directions. The equations of each minor step include the first and second partial derivatives of unknowns with respect to only one spatial variable. Thus, the performance of one minor step results in an independent solution for N x N one-dimensional boundary value problems.

The solution employs an implicit difference scheme with seven-point spatial pattern, each minor step comes down to solving between one and five systems of linear equations with three-diagonal matrix:

$$\begin{aligned} B_1 Y_1 + C_1 Y_2 &= F_1 , \\ A_i Y_{i-1} + B_i Y_i + C_i Y_{i+1} &= F_i , \quad i = 2,\ldots,N-1, \\ A_N Y_{N-1} + B_N Y_N &= F_N . \end{aligned}$$

The coefficients and right-hand sides of these equations may be computed independently of each other and in parallel throughout the computational domain. Calculation of the coefficients and right-hand sides involves evaluation of the first and second derivatives of unknown variables. The three-diagonal systems of equations are solved by the method of a scalar run through in each minor step, and N^2 scalar passes can be accomplished independently of each other.

The structure of the processor imposes certain limits on the possible parallel execution of the algorithm. The limited volume of the vector memory cannot hold all problem data at the same time, and therefore the data are kept in a second-level memory, and only information pertaining to a certain layer of the calculation region is called into the vector memory. The vector processor carries out the calculation by layers. In Fig. 8, the cube represents the calculation domain, while the planes depicted therein represent data kept in the vector memory.

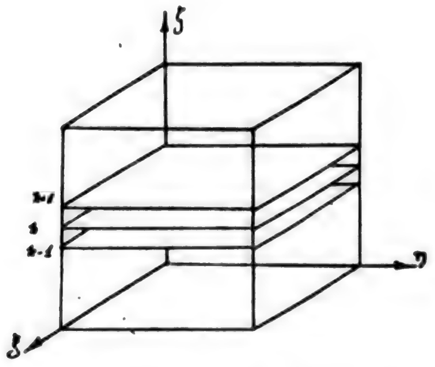

Fig. 8. Representation of data kept in the vector memory.

Say the first minor step involves derivatives of unknowns with respect to τ, the second with respect to η, the third with respect to ζ, the fourth again with respect to ξ, the fifth with respect to η, and the sixth with respect to ζ. Then, using data of the k-1 and k+1 planes, the coefficients and right-hand sides of the three-diagonal systems of the first minor step are evaluated for the k-th plane. These computations require three contiguous parallel planes, since the partial derivatives of first and second order are evaluated not only from ξ and η, but also from the normal, i.e., along ζ.

In order to evaluate the coefficients and right-hand sides of the equations of the first minor step, a pipeline processing is done for the matrix of the k-th plane. The three-diagonal system is solved by the economical method of a scalar run through. In the k-th plane, 50 scalar passes along ξ are required.

An individual scalar pass is poorly suited to parallel pipeline processing, since both the forward pass:

$$\alpha_{i+1} = \frac{-C_i}{A_i \alpha_i + B_i}, \quad \beta_{i+1} = \frac{F_i - A_i \beta_i}{A_i \alpha_i + B_i},$$

and the backward pass:

$$Y_{i-1} = \alpha_i Y_i + \beta_i$$

use recurrent formulas for the calculation.

However, the simultaneous execution of many (15) independent passes allows the use of parallel pipeline processing.

Thus, a "vector" of scalar passes along ξ (first forward, then backward) is accomplished. With this, the calculation of the first minor step is finished.

The calculation of the second minor step is done along η in the very same k-th plane. As in the first minor step, the coefficients are evaluated, and then a pass is made along η.

By adding information from the following k + 2 plane to the vector memory, we may perform the calculation of the first and second minor steps for the next k + 1 plane. Moving from one plane to another, the calculation of the first and second minor steps is performed for the entire calculation domain.

A different technique of parallel processing is used for the third minor step along ζ (the normal to $\xi\eta$).

In moving from the k-th to the k + 1-th plane during calculation of the first and second minor steps, the coefficients are evaluated for all 50 x 50 systems of equations of the third minor step and the k-th recurrent term is evaluated for each of the 50 x 50 forward passes along ζ. All these computations for the k-th plane are done in pipeline mode. Thus, after completion of the first and second minor steps in the entire calculation domain, it turns out that a "matrix" of straight passes along ζ has been accomplished at the same time. Moving through the planes in the backward direction, it is possible to accomplish a backward pass and complete the third minor step in the entire domain.

In the programming, the completion of the third minor step in the k-th plane is combined with the performance of the fourth and fifth minor steps in this plane. Furthermore, while moving from plane to plane during performance of the fourth and fifth minor steps the coefficients of the equations of the sixth minor step are evaluated and the k-th recurrent term of all 50 x 50 forward passes of the sixth minor step is evaluated. The final pass along ζ completes the sixth minor step and the entire time step as a whole.

More careful examination of the computation process reveals that the vector memory must simultaneously hold the data for not three, but four planes of the calculation domain. Furthermore, a buffer exchange zone should be set aside in the vector memory for the data of one plane.

The exchange of data with the Elbrus MCC should be done in arrays or matrices corresponding to the next plane of the calculation space. Each node of the grid during the process of the computation is described by 28 parameters. These include the density ρ, temperature T, components of the velocity u, v, w, increments of these quantities to be evaluated in the given time step, ξ_ρ, ξ_T, ξ_u, ξ_v, ξ_w, the length of the radius-vector r, the thermal conductance λ, the viscosity μ, the coefficients of transformation of the coordinates z_1, z_2, z_3, y_2, y_3 and the run through coefficients α_{20}, α_u, α_v, α_w, α_T, β_ρ, β_u, β_v, β_w, β_T. Thus, the data of the problem present a four-dimensional array of 50 x 50 x 50 x 28.

During the computation, four three-dimensional arrays corresponding to four planes, i.e., a dimension of 50 x 50 x 28, and several working arrays of dimension 50 x 50 are present in the vector memory. The problem is programmed in the underlying high-level language. The features of the programming language make it possible to access the required subarrays in the vector memory. The program has a modular structure and contains three basic modules.

The first module involves calculation in the current plane of the first and second minor steps, as well as the coefficients and evaluation of the next recurrent term of the third minor step in the entire plane. The first module also contains exchange instructions which send the results of the computation for the previous plane to memory and prepare the data for the following plane. The module has a rather large amount of computation and not many exchanges. The exchange is entirely integrated with the computation.

The second module completes the third minor step, includes the calculation of the fourth and fifth minor steps, and also contains a calculation of the next recurrent term of the sixth minor step for the entire current plane. During its operation, the module performs an exchange with the memory of Elbrus-2. The module has a rather large number of exchanges and the computation is not fully integrated with the exchange.

The third module completes the sixth minor step and the entire time step as a whole. There are few computations, and the operating time of the module is determined by the exchange.

The program of the three modules numbers 819 vector assignment operators, corresponding to approximately 2700 ordinary arithmetic scalar floating decimal operations per node in one time step. The volume of computations of the entire problem amounts to 10^{11} operations.

The program, the electronic componentry, and the thoroughly elaborated structural scheme of the vector processor enable certain quantitative assessments of its performance.

In the first place, we may estimate the time to perform any given stage of the computations: the time to perform one vector assignment operator is equal to the product of the duration of the clock period of the pipeline and the number of elements of the array (in our problem, the number of elements is 50 x 50 = 2500, and the startup time can be ignored). The time for an exchange is estimated by the ratio of the volume of information transmitted and the handling capacity of the exchange channel. Information on the modules (for a simultaneous step) of the program solving the Navier-Stokes equations is presented in Table 1 for a clock period of duration 44 ns and a rate of exchange of four million words per second. Thus, the problem including 300 time steps will be executed in 27 min.

Table 1

Номер модуля (a)	Число векторных операторов (b)	Время счета (мс) (c)	Число обменов (d)	Время обмена (мс) (e)	Время работы модуля (мс) (f)
I	667	3670	28	880	3670
2	I42	780	34	II70	II70
3	I0	55	I4	440	440
Итого (g)	8I9	4505	76	2490	5280

Key:

a.	Number or module	e.	Exchange time (ms)
b.	Number of vector operators	f.	Time of operation of module (ms)
c.	Calculation time (ms)	g.	Total
d.	Number of exchanges		

Secondly, we may obtain an estimate of the minimum volume of the vector memory needed to implement the described program. The vector memory should hold at least the data of five planes of the calculation space, i.e., 5 x 50 x 50 x 28 = 350 10^3 words. Such volume allows a run through in two orthogonal directions of the plane without supplementary accessing of the external memory. If a sufficiently fast vector memory is used, the speed of the process in this case is determined by the pace of operation of the processing devices. With a vector memory of lesser volume, the speed of the pipeline begins to depend explicitly on the rate of exchange (E_2) and declines sharply. In other words, the vector memory should make possible a reduction of the pace of accessing of the outer memory, i.e., the memory of Elbrus-2 MCC, while holding the data for the entire problem, in the amount of 3.6 million words, in the working storage of the Elbrus-2 MCC. A comparative evaluation of the speed of the BESM-6 and the vector processor in solving the Navier-Stokes equations is illustrated in Table 2.

Table 2

Parameter Compared	BESM-6	Vector Processor
Dimensions of grid	10 x 10 x 15	50 x 50 x 50
Volume of problem data (in numbers)	3×10^4	3.5×10^6
Problem solving time (in hours)	1	0.5
Relative speed (compared to BESM-6)	1	200

The program developed to solve the Navier-Stokes equations made it possible to evaluate the efficiency of utilization of the vector processor equipment. In particular, it was found that 3.34 of the five functional floating-decimal devices (two adders, two multipliers, and one divider) work on average, putting forth an average speed of approximately 80 MFLOPS. A histogram of the relative frequency of appearance of configurations with different numbers of functional devices is shown in Fig. 9.

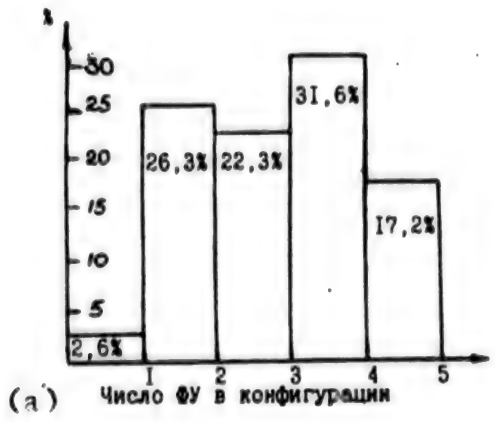

Fig. 9. Histogram of relative frequency of appearance of configurations (in percent) with differing number of functional devices.

Key:

a. Number of FD in configuration

It is interesting to observe the relatively high workload of the division device: it occupies 38.4 percent of the computation time. This is explained by the fact that, even though division comprises about 10 percent of the arithmetic operations in the present problem (as in many scientific and engineering problems), the simultaneous operation of the two adders and two multipliers almost quadruples the frequency of access to the divider.

The obtained results reveal an excellent and rather well-balanced workload of the functional devices, testifying to the rational choice of the outfit of functional devices.

Bibliography

1. Burtsev, V. S., "Tendentsii razvitiya vysokoproizvoditelnykh sistem i mnogoprotsessornyye vychislitelnyye kompleksy" [Development Trends of High Performance Systems and Multiprocessor Computing Complexes], M., 1977, 28 pp. (Preprint ITMiVT).

2. Burtsev, V. S., "Printsipy postroyeniya mnogoprotsessornykh vychislitelnykh kompleksov 'Elbrus'" [Principles of Design of the Elbrus Multiprocessor Computing Complexes], M., 1977, 53 pp., (Preprint 1, ITMiVT).

3. Burtsev, V. S., "Analiz rezultatov ispytaniy MVK 'Elbrus-2' i dalneyshiye puti yego razvitiya" [Analysis of Results of Testing the Elbrus-2 MCC and Further Avenues of Its Development], M., 1988, (Preprint 208, OVM AN SSSR).

4. Burtsev, V. S., Krivosheyev, Ye. A., Asriyeli, V. D., et al., "A Vector Processor with Programmable Structure," in "Vychislitelnyye protsessy i sistemy" [Computing Processes and Systems], Issue 2, Nauka, M., 1985.

5. Asriyeli, V. D., Myasnikov, A. V., Popov, T. A., "Experience in Creation of the Vector Processor Software," (Present Volume).

6. Asriyeli, V. D., Popov, T. A., "MASSIV, the Underlying Programming Language of the Vector Processor," (Present Volume).

7. Asriyeli, V. D., Borisov, P. V., "Experience in Programming for the Vector Processor to Solve the Navier-Stokes Equations in a Three-Dimensional Domain," in "Vychislitelnyye protsessy i sistemy," [Computing Processes and Systems], Issue 2, Nauka, M., 1985.

8. Kovenya, V. M., Yanenko, N. N., "Metod rasshchepleniya v zadachakh gazovoy dinamiki" [The Method of Partitioning in Gas Dynamic Problems], Nauka, Novosibirsk, 1981.

9. Kovenya, V. M., Lebedev, A. S., "Chislennoye resheniye uravneniy Navye-Stoksa dlya techeniya nad bokovoy poverkhnostyu tela i v slede za nim" [Numerical Solution of the Navier-Stokes Equations for Flow Above the Lateral Surface of a Solid and in the Wake Behind It], ITPM SO AN SSSR, Novosibirsk, 1982.

10. Olenin, A. S., "Several Structural and Algorithmic Aspects of the Scalar-Vector Model of Computation," (Present Volume).

11. Olenin, A. S., "Skalyarno-vektornyye protsessy v lentochnykh sistemakh" [Scalar-Vector Processes in Banded Systems], M., 1987, 29 pp. (Preprint 156, OVM AN SSSR).

12. Fetisov, N. S., Tverdokhlebov, M. V., Sablukov, A. A., Kramarenko, A. N., "Architecture and Organization of the Diagnostic Utility of the E-2 MCC Vector Processor," (Present Volume).

13. Kascis, M. J., "Vector Processing on the Cyber 205," "Infotech State of the Art Report Supercomputers," Vol 2, 1979, pp. 237-270.

14. Russel, R. M., "The Cray-1 Computer System," COMM. ACM, Vol 21, No 1, 1978, pp. 63-72.

15. Thompson, J. R., "The Cray-1, the Cray X-MP, the Cray-2 and Beyond: The Supercomputers Class VI Systems, Hardware and Software," North Holland, 1986, pp. 69-81.

16. "CDC Cyber-200 Model 205, Technical Description," Control Data Corporation, November 1980.

MASSIV, THE UNDERLYING PROGRAMMING LANGUAGE OF THE VECTOR PROCESSOR

907G0089B Moscow SUPER-EVM in Russian, pp. 30-48

[Article by V. D. Asriyeli and T. A. Popov]

[Text] The vector processor (VP) of the Elbrus multiprocessor computing complex (MCC) is designed for effective solving of a broad class of scientific and engineering problems [1]. An underlying high-level programming language was specially developed for the VP, enabling maximum effectiveness in use of the VP [2]. The present work contains a complete description of the realized version of the underlying language of the VP of the Elbrus MCC.

The underlying programming language of the VP is oriented to working with arrays. The underlying language can describe arrays with dimensionality not greater than four, although only one-dimensional or two-dimensional arrays may take part in the operations. The underlying language allows four types of variables:

- integer (32-bit signed integer);

- scalar (64-bit real-valued number);

- array (consisting of 64-bit real-valued numbers);

- external array (located in the second-level memory).

The underlying language is based on a consecutive operator scheme of programs with parallel assignment operator of array type (AOAT). The semantics of the AOAT distinguishes this scheme from that underlying the majority of presentday programming languages (ALGOL, FORTRAN, etc.). The AOAT is defined as a compact notation of a set of elementary assignment operators (EAO), defined on the analogous components of the arrays, and the use of the AOAT is defined as an asynchronous parallel execution of the constituent EAOs. The AOAT is considered to be executed when all its EAOs have been executed.

Descriptions

The unit of programming (and translation) of the underlying language is the module. Of the modules comprising the program package, the module which begins the running of the program is singled out. This head module (program) differs from an ordinary module in the fact that it contains an external memory description partition instead of a parameter partition. The external memory partition is used to map the external arrays on the second-level memory

(RAM of the Elbrus MCC). The initial contents of the second-level memory are formed by external means (relative to the VP), i.e., the resources of the Elbrus MCC.

A module in the underlying language consists of descriptions and operators. The descriptions are divided into five partitions:

- the partition of formal parameters;

- the partition of local variables;

- the partition of subarrays;

- the partition of invoked modules;

- the partition of macrodefinitions.

The formal parameter partition is designed to describe the parameters of the module. The information contained in this partition is used in particular by the compiler in compiling the object modules. A module may have formal parameters of all four types, which receive the values of the actual parameters when the module is called up; the parameters for variables of integer and scalar type are given by a value, those for variables of array and external array type are given by an access (i.e., a formal array-parameter gives the module access to an actual array-parameter).

Let us consider the descriptions of the arrays more closely. The underlying language defines two similar types of variables: the array and the external array. These differ only in the fact that:

- the array is situated in the RAM, while the external array is in the second-level memory (RAM of the MCC);

- the module may describe a local variable of array type, but not a local variable of external array type;

- only variables of array type may participate in the operations, while the variables of external array type may occur only in exchange operators.

In each measurement, the elements of an array (and an external array) are numbered with nonnegative whole numbers, starting at zero. Thus, if the quantity of elements in a certain measurement is k, the elements in this measurement are numbered from 0 to k-1.

The local variable partition describes variables which are created when the module is called up and cancelled upon return from the module. All descriptors of measurements in the description of a local array are of the kind <DESCRIPTIVE EXPRESSION>, where

```
<DESCRIPTIVE EXPRESSION> :: = <NAME OF ARRAY>.<NUMBER OF MEASUREMENT>|
                              <D.NUMBER>|
      <DESCRIPTIVE EXPRESSION><+-><DESCRIPTIVE EXPRESSION>
<NUMBER OF MEASUREMENT> :: =   1|2|3|4
<+-> :: = +|-
```

The value of the descriptive expression determines the number of elements in the given measurement of the described array. If the descriptive expression is <NAME OF ARRAY>.<NUMBER OF MEASUREMENT>, the corresponding quantities of elements are linked by an equivalence relation (i.e., they occupy the same registers of the register file). Thus, e.g., the description (Fig. 1):

/M2/ D[B.1, B.2-2]

defines a two-dimensional array, the number of rows in which coincides with the quantity of elements for the first measurement of the array B, while the quantity of columns is 2 less than the quantity of columns of array B. The quantity of elements in the first measurement of array D and the quantity of elements in the first measurement of array B are linked by an equivalence relation.

```
%МОДУЛЬ(ПРИМЕР)  (1)
%ПАРАМЕТРЫ      (2)
    /M2/ A,B,C
%ЛОКАЛЬНЫЕ       (3)
    /Ц/  N
    /M2/ D[B.1,B.2-2]
    /M2/ E[B.1,C.2]
    /MI/ F[B.1]          (4)
%ПОДМАССИВЫ
    /MI/ G[1] = B[*,5]
    /ДМI/ H[1](N) = B[*,N]
%МОДУЛИ                    (5)
    /МОД/ ПРИМЕРI(B,D,H)     (6)
%ОПЕРАТОРЫ                 (7)
    . . . . .
    ПРИМЕРI(C,D,G)
    . . . . .              (8)
    ПРИМЕРI(C,D,F)

    . . . . .
%КОНЕЦ               (9)
```

Fig. 1

Key:

1.	Module (example)	6.	/Mode/Example 1
2.	Parameters	7.	Operators
3.	Local	8.	Example 1
4.	Subarrays	9.	End
5.	Modules		

The subarray partition describes subarrays of previously described arrays (reference arrays). There are two common types of subarray: static and dynamic. Only a formal array, a local array, or a static subarray can be a reference array.

A static subarray is defined once when the module is called up and cannot be redefined during the execution of the module. The description of a dynamic subarray uses an integer-type associated variable, and the dynamic subarray is redefined with each change in value of the associated variable (Fig. 1).

In the left part of the description of a static subarray, in place of the determinants of measurements there appear the numbers of measurements of the reference array, providing the orientation for the measurements being described. The measurement numbers will be different in different measurements of the described array. In the right part of the description appears the name of the reference array, while the square brackets contain clarifying information on each measurement of the reference array, which along with the left part of the description uniquely determines the array being described. The clarifying information is of the type:

- either *, when the measurement of the described array referring to a given measurement of the reference array coincides with the given measurement;

- or <START>:<QUANTITY>, when the measurement of the described array referring to a given measurement of the reference array is a contraction of the given measurement. If the quantity is of the type <NAME OF ARRAY>.<NUMBER OF MEASUREMENT>, then the determined quantity of the described array is linked by an equivalence relation to the corresponding quantity of the previously described array;

- or <DESCRIPTIVE EXPRESSION>, when there is no reference to a given measurement of a reference array in the array being described, i.e., the described array has lesser dimensionality.

<START> :: = <DESCRIPTIVE EXPRESSION>
<QUANTITY> :: = <DESCRIPTIVE EXPRESSION>

For example, the description

/M2/ A[3, 2] = B[5, *, 1:B.3-2]

describes a two-dimensional subarray A of a three-dimensional array B. In array B, the fifth slice is taken for the first measurement. In the resulting two-dimensional array, the first and last columns are discarded and the remaining array is transposed. The array resulting from all of these manipulations is the subarray A. Of course, no displacement, discarding, or transposition of the elements of array B takes place. Array A simply obtains access to those elements of array B that would remain if the above transformations were performed. A further result of such description of array A is that the quantity of elements for the second measurement of array A is linked by an equivalence relation with the quantity of elements for the second measurement of array B.

On the left side of a description of a dynamic subarray, in place of the measurement descriptors appear the numbers of the measurements of the reference array, providing an orientation for the measurements being described. The numbers of the measurements will be different in different measurements of the described array. Next, in round brackets, comes the name of the associated integer variable, the equal sign, the name of the reference array and, in square brackets, the determining information. The dimensionality of a dynamic subarray is one unit less than that of the reference array, which is the reason for the existence of the number of the measurement of the reference array, which is not found on the left side of the description of the dynamic subarray. In place of this measurement of the reference array appears the name of the associated integer variable (the same one), while in place of the other measurements of the reference array appears the symbol *.

Example:

/ДМ2/ A[2,1] (HOMEP) = B[•,•,HOMEP(a)]

Key:

a. Number

The partition of invoked modules describes the structure of formal parameters of the modules being called up. The information of this partition is used as a compiler in compiling the object modules. The structure of the actual parameters in the module call operator should agree with the structure of the formal parameters specified in the invoked module partition.

Operators

Five basic kinds of operator exist in the underlying language:

- the assignment operator;

- the module call operator;

- the return from module operator;

- the branch operator;

- the exchange operator.

In accordance with the types of variables appearing in the assignment operator, we distinguish three types of assignment operator:

- the integer type assignment operator;

- the scalar type assignment operator;

- the array type assignment operator.

Furthermore, the language has a construct which combines two array type assignment operators, enabling simultaneous execution of both operators. This construct is known as the DOUBLET. For example, simultaneous execution of two assignment operators of array type:

X: = Y+Z
Q: = (Q+R+S)/T,

where X, Y, Z, Q, R, S, T are one-dimensional or two-dimensional arrays, is specified by a single DOUBLET construct:

DOUBLET (X: = Y+Z, Q: = (Q+R+S)/T)

The DOUBLET construct is defined as a compact notation of a set of EAOs, obtained by unifying the sets of EAOs of two AOATs. The execution of this construct is defined as an asynchronous parallel execution of the EAOs of a unified set.

The right side of the assignment operator is an expression of appropriate type. Different groupings of two-place operations are generally involved in the formation of expressions of different types. The grouping of operations of integer type includes:

+ - the operation of integer addition;

- - the operation of integer subtraction;

* - the operation of integer multiplication;

& - the operation of bitwise logical multiplication;

% - the operation of bitwise logical addition;

- the operation of bitwise logical comparison;

<> - the operation of bitwise logical shifting.

In the shift operation, the number being shifted is the first operand, while the magnitude and direction of the shift are the second operand (negative value means shift to left).

The grouping of scalar type and array type operations includes:

+, -, * - addition, subtraction, multiplication;

/ - division;

>< - cutoff maximum (A><B means MAX (A-B,0));

&, %, # - bitwise logic operations;

<> - shift.

Furthermore, three functions of a single argument may be involved in forming a scalar type expression: SQUARE, DELTA, and INT.PART [KVARDAT, DELTA, and TsChAST]. The result of the function DELTA is a real-valued unit, if the argument is less than zero, and a real-valued zero otherwise. The result of the function INT.PART is the closest integer to the argument, the modulus of which does not exceed the modulus of the argument, and the resulting integer is represented as a normalized real-valued number. The function SQUARE enables a highly effective (without using intermediate variables) computation of the square of a value.

Three functions of a single argument may be involved in forming array type expressions: DELTA, INT.PART, and ARR [DELTA, TsChAST, and MAS]. The argument and result of the DELTA (INT.PART) function is an array, the result being a componentwise application of the scalar function DELTA (INT.PART) to the elements of the argument-array. The argument of the function ARR is a scalar, and the result is an array, the structure of which depends on the context (since the operations in the expression are performed on arrays with identical structure).

All the operations and functions involved in formation of array type expressions are divided into groups:

- multiplication (*);

- addition (+, -, ><, INT.PART);

- division (/);

- logic (DELTA, &, %, #, <>);

- scalar (ARR).

The underlying language places certain restrictions on the form of the array type assignment operator and the DOUBLET construct. In one assignment operator (or two operators of the DOUBLET construct) there should be:

- not more than two operations of the multiplication group;

- not more than two operations of the addition group;

- not more than one operation of the division group;

- not more than one operation of the logic group;

- not more than four operations of the scalar group;

- not more than six entries of names of arrays in the right-hand part of the operator (right-hand sides of the DOUBLET construct).

By operation here is meant both a two-place operation and a function of one argument.

Expressions are evaluated in accordance with the priorities. The highest priority goes to functions of one argument and expressions enclosed in brackets (priority 0). Two-place operators have the following priorities:

& - priority 1;

% - priority 2;

# - priority 3;

<> - priority 4;

* - priority 5;

/ - priority 5;

+ - priority 6;

- - priority 6;

>< - priority 6.

The lowest priority (7) is of the group of relation operations:

= - equals;

/= - not equal;

< - less than;

<= - less than or equal to;

> - greater than;

>= - greater than or equal to.

Relation operations may occur only in a conditional jump operator and an exchange operator.

All operations on arrays are performed componentwise, i.e., an analogous scalar operation is applied to the like components of the arrays. In an array type assignment type operator (and DOUBLET construct), all arrays should have identical dimensionality and identical quantities of elements in the like measurements.

The module call operator contains factual parameters, and the structure of the factual parameters should agree with that of the formal parameters of the called module. A recursive module call is permitted. The return from module operator returns control to the calling module.

The branch operators are divided into conditional and unconditional. The conditional branch operator passes control to a label within the given module if the condition is true and passes control to the next operator if the condition is false. There are two types of condition: integer and scalar.

The exchange operator has two modifications (input and output). The input operator assigns to an array type variable the value of an external array type variable. The output operator assigns to an external array type variable the value of an array type variable. If the condition field in the exchange operator is not empty, the operator is performed when the condition is true. If the condition is false, the operator is not performed (it is passed over). An exchange may specify recoding into the format of the Elbrus CP (E) or the Elbrus SVS (B).

Exchange operators are performed in strict sequence, i.e., the next exchange operator will be performed only after total completion of the exchange of the previous exchange operator. An exchange operator initiates the current exchange and, without waiting for total completion of the current exchange, passes control to the next operator. The operator SYNCHRO is used to synchronize the exchange and the computations, guaranteeing that the next operator will be initiated only after total completion of the current exchange.

Additional Capabilities

The operator SPEC1 makes it possible to form a 64-bit scalar value from 32-bit integer values. The first integer expression forms the high-order bits of the scalar, the second the low-order bits. The operator SPEC2 makes it possible to slice a 64-bit scalar value into two 32-bit elements and to store them in integer variables. The first integer variable is assigned the value of the high-order bits of the scalar, the second the value of the low-order bits.

The integer function of two arguments INV performs a double inversion of the value of the first argument and then shifts the result to the right by the number of bits indicated in the second argument.

For array type variables, along with the DOUBLET construct there is yet another construct (FORK), which in certain cases can boost the efficiency of the program. This construct is subject to the same quantitative and qualitative restrictions on the makeup of the operations as the DOUBLET construct and the array type assignment operator. The efficiency is enhanced by using common subexpressions within a single FORK construct. The FORK construct contains one or two array type assignment operators. In the case of two operators (as in the DOUBLET construct), both operators will be performed at the same time. However, unlike the DOUBLET construct, both operators may contain common subexpressions that are evaluated once. The array type expression is specially amplified for the FORK construct. Thus, notations of type:

```
[<DIGIT><A.EXPRESSION>] and [<DIGIT>]
```

are array type expressions, the first notation defining a common subexpression and assigning to this subexpression a number, while the second notation is a reference to the defined subexpression. For example, two assignment operators of array type:

```
F : = A/((B+C)*(C-D)),
G : = (B+C)*(C-D)*E
```

can be replaced with a single FORK construct:

```
FORK (F : = A/[1(B+C)*(C-D)], G : = [1]*E)
```

or

```
FORK (F : = A/[1(B+[2C])*([2]-D)], G : = [1]*E),
```

the latter construct being preferable, since (other conditions being equal) there are fewer names of arrays on the right side of the assignments. The first FORK construct corresponds to the processing pipeline configuration shown in Fig. 2, the second FORK construct corresponds to the processing pipelire configuration in Fig. 3.

The digits in the square brackets of the FORK construct project the points of branching. The second FORK construct has an important attribute: two different paths, issuing from the same branching point (number 2), converge on the same functional device (multiplier), forming a so-called contour. All contours of the processing pipeline should be balanced, i.e., both paths forming a contour should have the same length. The path length is equal to the sum of delays of the functional devices through which the path passes. A delay line, which may have a length of 8, 16, 24, 32 (respectively 1, 2, 3, 4 in the underlying language), is used to equalize the path length. In the language, a delay line corresponds to a function of two arguments DELAY (3). The first argument of the delay-function is the magnitude of the delay, while the second argument is an array type expression, the values of which are "delayed." The delay-function is part of the group of logic operations. Since all functional devices (except the divider) have a delay of 8 (the divider has 16), two array type assignment operators:

```
G : = A/B*(B+C),
H : = (D+E)*F
```

are equivalent to the FORK construct:

```
FORK (G : = A/[1B]*3(1,[1]+C),H: = (D+E)*F),
```

it being impossible in this case to forego a contour (i.e., by using a DOUBLET construct), since more than six names of arrays would then appear on the right sides. This FORK construct corresponds to the processing pipeline configuration shown in Fig. 4. The semantics of the FORK construct is analogous to the semantics of the array type assignment operator (AOAT) and the DOUBLET construct.

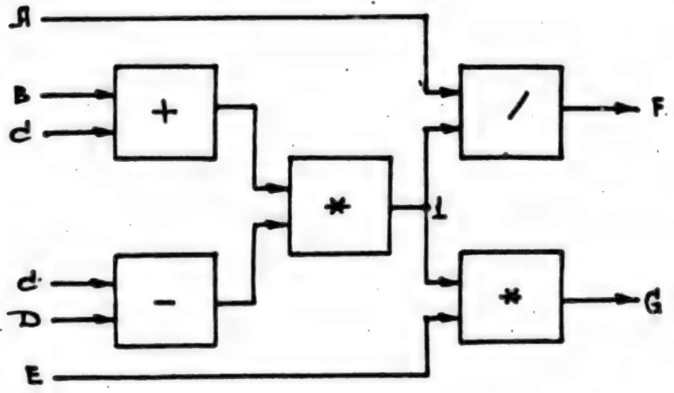

Fig. 2

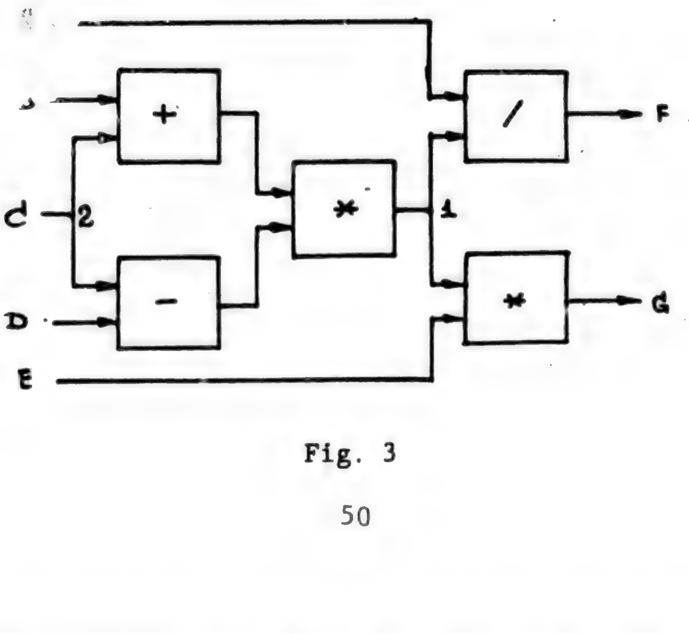

Fig. 3

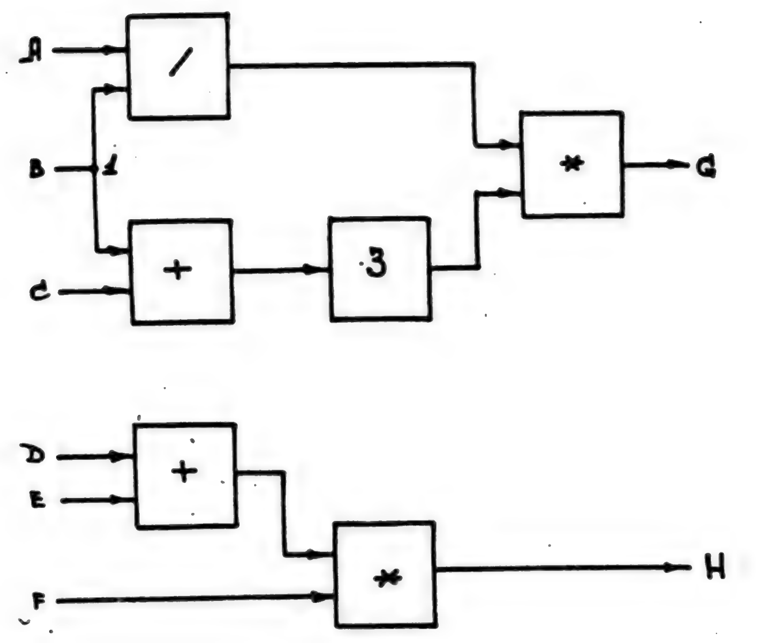

Fig. 4

The CONVOLUTION operator makes it possible to convolute the array result of evaluation of a given expression by a single operation (+, *, %, #, &), writing the result as a scalar variable. For example, the operator:

```
CONVOLUTION (C: = +[A*B])
```

will write the value of the scalar product of vectors A and B into the scalar variable C.

Not more than one operation of the logic group may appear on the right side of the operator CONVOLUTION.

Syntax

In describing the syntax, an effort is made to portray the possible arrangement of the described constructs in the lines of a program. Therefore, if a certain alternative occupies several lines, this means that the concepts of which it is formed should begin with a new line. An obvious exception to this rule is when the text of the alternative does not fit on one line.

An identifier is a sequence of letters or digits, beginning with a letter. The length of an identifier is not more than eight characters.

```
<PROGRAM> : : = %PROGRAM(<IDENTIFIER>)
                  <PARTITION EX. MEMORY>
                  <DESCRIPTIONS>
                  <PARTITION OF OPERATORS>
                  %END
<MODULE> : : = %MODULE(<IDENTIFIER>)
                  <PARTITION OF PARAMETERS>
                  <DESCRIPTIONS>
```

```
                    <PARTITION OF OPERATORS>
                    %END
<DESCRIPTIONS> :: = <PARTITION OF LOCALS>
                        <PARTITION OF SUBARRAYS>
                        <PARTITION OF MODULES>
                        <PARTITION OF MACRODEFINITIONS>
<PARTITION OF PARAMETERS> :: = %PARAMETERS
                                <FORM.PARAMETERS>|
<FORM.PARAMETERS> :: = <FORM.PARAMETER>|
                            <FORM.PARAMETER>.<FORM.PARAMETERS>|
                            <FORM.PARAMETERS>
                            <FORM.PARAMETERS>
<FORM.PARAMETER> :: = <INT.DESCRIPTION>|<AR.DESCRIPTION>|
                        <S.DESCRIPTION>
<INT.DESCRIPTION> :: = /INT/ <LIST OF IDENTIFIERS>
<LIST OF IDENTIFIERS> :: = <IDENTIFIER>|
                    <IDENTIFIER>,<LIST OF IDENTIFIERS>
<AR.DESCRIPTION> :: = <TYPE OF ARRAY><LIST OF IDENTIFIERS>
<TYPE OF ARRAY> :: = /AR<DIMENSION>/|/EX.AR<DIMENSION>/
<DIMENSION> :: = 1 | 2 | 3 | 4
<S.DESCRIPTION> :: = /S/<LIST OF IDENTIFIERS>
<PARTITION OF LOCALS> :: = %LOCAL
                            <LOCAL VARIABLES>|
<LOCAL VARIABLES> :: = /INT/<INT.C.DESCRIPTION>|
                        /S/<S.C.DESCRIPTION>|
                        <INT.DESCRIPTION>|<S.DESCRIPTION>|
                        <L.AR.DESCRIPTION>|
                        <LOCAL VARIABLES>,<LOCAL VARIABLES>|
                        <LOCAL VARIABLES>
                        <LOCAL VARIABLES>
<INT.C.DESCRIPTION> :: = <IDENTIFIER> = <INT.CONSTANT>|
                        <IDENTIFIER> = <INT.CONSTANT>,<INT.C.DESCRIPTION>
<INT.CONSTANT> :: = <D.NUMBER>|'<H.NUMBER>'|-<D.NUMBER>
<D.NUMBER> :: = <DIGIT>|<DIGIT><D.NUMBER>
<H.NUMBER> :: = <H.DIGIT>|<H.DIGIT><H.NUMBER>
<DIGIT> :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<H.DIGIT> :: = <DIGIT> A | B | C | D | E | F
<S.C.DESCRIPTION> :: = <IDENTIFIER> = <S.CONSTANT>|
                        <IDENTIFIER> = <S.CONSTANT>,<S.C.DESCRIPTION>
<S.CONSTANT> :: = <D.NUMBER>.<D.NUMBER><EXPONENT>|
                    -<D.NUMBER>.<D.NUMBER><EXPONENT>|
                    '<H.NUMBER>'
<EXPONENT> :: = | e<+-><D.NUMBER>
<L.AR.DESCRIPTION> :: = /AR<DIMENSION>/<LIST LOC.AR.>
<LIST LOC.AR.> :: = <LOC.ARRAY.|
                        <LOC.ARRAY>,<LIST LOC.AR.>
<LOC.ARRAY> - see "descriptions" section
<PARTITION OF SUBARRAYS> :: = %SUBARRAYS
                            <DESC.SUBARRAYS>|
<DESC.SUBARRAYS> :: = <SUBARRAYS>|
                        <SUBARRAYS>,<DESC.SUBARRAYS>|
                        <DESC.SUBARRAYS>
                        <DESC.SUBARRAYS>
<SUBARRAYS> :: = <TYPE OF SUBARRAY><LIST OF DYN.SUBAR.>|
                  <TYPE OF SUBARRAY><LIST OF STAT.SUBAR.>
<LIST OF STAT.SUBAR.> :: = <STAT.SUBARRAY>|
                        <STAT.SUBARRAY>,<LIST STAT.SUBAR.>
<LIST DYN.SUBAR.> :: = <DYN.SUBARRAY>|
                        <DYN.SUBARRAY>,<LIST DYN.SUBAR.>
<PARTITION OF MODULES> :: = %MODULES
                              <DESCRIPTIONS OF MODULES>|
<DYN.SUBARRAY> - see "descriptions" section.
<STAT.SUBARRAY> - see "descriptions" section.
<DESCRIPTIONS OF MODULES> :: = <DESCRIPTION OF MODULE>|
                                  <DESCRIPTION OF MODULE>
                                  <DESCRIPTION OF MODULES>
<DESCRIPTION OF MODULE> :: = /MOD/<IDENTIFIER> (<PARAMETERS>)
<TYPE OF SUBARRAY> :: = /D.AR.<DIMENSION>/|
                        /D.EX.AR.<DIMENSION>/|<TYPE OF ARRAY>
<PARAMETERS> :: = <PARAMETERS>|<PARAMETER>,<PARAMETERS>
<PARAMETER> :: = <NAME OF INTEGER>|<NAME OF SCALAR>|
                  <NAME OF INT.CONSTANT>|<NAME OF S.CONSTANT>|
                  <NAME OF ARRAY>|<NAME OF EXTERNAL ARRAY>
<PARTITION OF OPERATORS> :: = %OPERATORS
                              <OPERATORS>
<OPERATORS> :: = <LABELED OPERATOR>
                  <OPERATORS>|
<LABELED OPERATOR> :: = <LABEL> : <OPERATOR>|<OPERATOR>
<LABEL> :: = <IDENTIFIER>
<OPERATOR> :: = <ASSIGNMENT>|<CALL>|<RETURN>|
                  <BRANCH>|<EXCHANGE>|<DOUBLET>|<SYNCHRO>|
                  <FORK>|<CONVOLUTION>|<SPEC.OPERATOR>|
                  <MACROCALLS>
<ASSIGNMENT> :: = <INT.ASSIGNMENTS>|
                  <S.ASSIGNMENTS>|
                  <AR.ASSIGNMENT>
<INT.ASSIGNMENTS> :: = <INT.ASSIGNMENT>,<INT.ASSIGNMENTS>|
                        <INT.ASSIGNMENT>
<INT.ASSIGNMENT> :: = <NAME OF INTEGER>:=<INT.EXPRESSION>
<CONVOLUTION> :: = CONVOLUTION (<NAME OF SCALAR>:=<OPER.>[<CONV.EXP.>])
<OPER.> :: = + | * | % | # | &
<CONV.EXP.> :: = <NAME OF ARRAY>|
                  <NAME OF ARRAY><AR.OPERATION><NAME OF ARRAY>
<INT.EXPRESSION> :: = <NAME OF INTEGER>|<NAME OF INT.CONSTANT>|
                        <INT.CONSTANT>|(<INT.EXPRESSION>)|
                        <NAME OF ARRAY>.<NUMBER OF MEASUREMENT>|
                        <INT.EXPRESSION><INT.OPERATION><INT.EXPRESSION>|
                        INV(<INT.EXPRESSION>,<D.NUMBER>)
<NUMBER OF MEASUREMENT> :: = 1 | 2 | 3 | 4
<INT.OPERATION> :: = + | - | * | & | % | # | <>
<S.ASSIGNMENTS> :: = <S.ASSIGNMENT>|
                        <S.ASSIGNMENT>,<S.ASSIGNMENTS>
<S.ASSIGNMENT> :: = <NAME OF SCALAR> :=<S.EXPRESSION>|
                        <VARIABLE WITH INDEXES> :=<S.EXPRESSION>
<VARIABLE WITH INDEXES> :: = <NAME OF ARRAY>[<INDEXES>]
<INDEXES> :: = <INT.EXPRESSION><INDEXES>|
                  <INT.EXPRESSION>
<S.EXPRESSION> :: = <NAME OF SCALAR>|<NAME OF S.CONSTANT>|
                         <VARIABLE WITH INDEXES>|
                         (<S.EXPRESSION>)|<S.FUNCTION>|
                         <S.EXPRESSION><S.OPERATION><S.EXPRESSION>
<S.OPERATION> :: = + | - | * | / | >< | & | % | # | <>
<S.FUNCTION> :: = DELTA(<S.EXPRESSION>)|
                  INT.PART(<S.EXPRESSION>)|
                  SQUARE(<S.EXPRESSION>)
<AR.ASSIGNMENT> :: = <NAME OF ARRAY> :=<AR.EXPRESSION>
<AR.EXPRESSION> :: = <NAME OF ARRAY>|<AR.FUNCTION>|
                         (<AR.EXPRESSION>)|
                         <AR.EXPRESSION><AR.OPERATION><AR.EXPRESSION>|
                         [<DIGIT><AR.EXPRESSION>]|[<DIGIT>]
<AR.FUNCTION> :: = DELTA(<AR.EXPRESSION>)|
                         INT.PART(<AR.EXPRESSION>)|
                         AR.(<NAME OF SCALAR>)|
                         DELAY(<D.NUMBER>,<AR.EXPRESSION>)
<CALL> :: = <NAME OF MODULE> (<F.PARAMETERS>)
<F.PARAMETERS> :: = <F.PARAMETER>,<F.PARAMETERS>|
                         <F.PARAMETER>
<F.PARAMETER> :: = <NAME OF ARRAY>|<INT.EXPRESSION>|
                         <NAME OF SCALAR>
<AR.OPERATION> :: = + | - | * | / | >< | & | % | # | <>
<RETURN> :: = RETURN
<GO TO> :: = CONTROL(<CONDITION>) <LABEL>
<CONDITION> :: = <INTEGER CONDITION>|<SCALAR CONDITION>|
<INTEGER CONDITION> :: = <INT.EXPRESSION><RELATION><INT.EXPRESSION>
<SCALAR CONDITION> :: = <NAME OF SCALAR><RELATION> 0
<RELATION> :: = < | > | = | >= | <= | /=
<EXCHANGE> :: = INPUT(<UO>)(<RECODE>)(<NAME OF ARRAY>,<NAME OF EXTERNAL
ARRAY>)
      <OUTPUT>(<UO>)(<RECODE>)(<NAME OF ARRAY>,<NAME OF EXTERNAL ARRAY>)
<DOUBLET> :: = DOUBLET(<AR.ASSIGNMENT>,<AR.ASSIGNMENT>)
<UO> :: = <CONDITION>
<RECODE> :: = E | B |
<SYNCHRO> :: = SYNCHRO
<FORK> :: = FORK(<AR.ASSIGNMENT>)|
            FORK(<AR.ASSIGNMENT>,<AR.ASSIGNMENT>)
<SPEC.OPERATOR> :: = SPEC1(<NAME OF SCALAR>:=<INT.EXPRESSION>,
<INT.EXPRESSION>)
                  SPEC2(<NAME OF INTEGER>,<NAME OF INTEGER>:=<NAME OF SCALAR>)
<PARTITION OF EX.MEMORY> :: = %MEMORY
                               <EX.ARRAYS>|
<EX.ARRAYS> :: = <DESCRIPTION EX.AR.>|
                  <DESCRIPTION EX.AR.>,<EX.ARRAYS>|
                  <EX.ARRAYS>
                  <EX.ARRAYS>
<DESCRIPTION EX.AR.> :: = /EX.AR.<DIMENSION>/<LIST LOC.AR.>
<PARTITION OF MACRODEFINITIONS> :: = %MACRODEFINITIONS
                                       <DEF.MACROBASED CROSSES>|
<DEF.MACROBASED CROSSES> :: = <DEF.MACROBASED CROSS>.
                               %
                               <DEF.MACROBASED CROSSES>|<DEF.MACROBASED CROSS>
<DEF.MACROBASED CROSS> :: = MACROBASED CROSS <NAME OF MACROBASED CROSS>
                               <TEXT OF MACRODEFINITION>
```

```
<NAME OF MACROBASED CROSS> :: = <IDENTIFIER>
<MACROCALL> :: = %<ARRAY/MACHINE CODE> MACROBASED CROSS <PRINT MODE>
                <NAME OF MACROBASED CROSS>(<F.PARAMETERS>)
<PRINT MODE> :: = P |
<ARRAY/MACHINE CODE> :: = | A
```

The parameters and local labels appearing in the text of the macrodefinition are signified respectively by: %<NUMBER OF PARAMETER> and ;<D.NUMBER>. The parameters are numbered from left to right (starting at one). The macrocall specifies the language in which the macrodefinition is written and indicates the necessity of printing out the text of the macrosubstitution.

The order of description of local variables and the order of variables in the list of parameters of a module are as follows:

1) scalars;

2) integers;

3) arrays.

An empty condition in the branch and exchange operators indicates unconditional execution of the given operators.

The powers of expression of the underlying language were tested in the programming of the solution of the Navier-Stokes equations [3]. Translators of the underlying language have been created in the BESM-6 and the Elbrus MCC.

Bibliography

1. Burtsev, V. S., et al., "A Vector Processor with Programmable Structure," VYCHISLITELNYYE PROTSESSY I SISTEMY, Nauka, M., 1985, Issue 2, pp. 63-72.

2. Asriyeli, V. D., "The Underlying Language of the Programming System of the Vector Processor," VYCHISLITELNYYE PROTSESSY I SISTEMY, Nauka, M., 1985, Issue 2, pp. 73-83.

3. Asriyeli, V. D., Borisov, P. V., "Experience in Programming the Solution of Navier-Stokes Equations in a Three-Dimensional Region for the Vector Processor," VYCHISLITELNYYE PROTSESSY I SISTEMY, Nauka, M., 1985, Issue 2, pp. 84-90.

EXPERIENCE IN CREATING THE SOFTWARE OF THE VECTOR PROCESSOR

907G0089C Moscow SUPER-EVM in Russian, pp. 49-66

[Article by V. D. Asriyeli, A. V. Myasnikov, and T. A. Popov]

[Text] Introduction

The creation of the software of the vector processor (VP) is part of a many year project, the goal of which was to develop a VP for the Elbrus-2 multiprocessor computing complex (MCC). The development of the VP for the Elbrus-2 MCC culminated in the creation of design documentation for factory manufacture, creation of a set of utilities for development of programs for the VP in the BESM-6, and implementation of the language components of the programming system in the Elbrus-2 MCC. The course of development of the VP, as well as several intermediate results, are described in part by the available publications [1-6].

The special nature of the VP software comes from the fact that the VP is not a stand-alone computing system, but a processor that is incorporated in place of one of the central processors in the Elbrus-2 MCC. Creation of a stand-alone system with new architecture requires development of a full volume of software and confronts the designers with the problem of transferring the existing application programs. Resolution of this problem largely determines the competitiveness of the new system. From this perspective, the MCC with VP has an indubitable advantage, since it does not require implementation of the full volume of software for the VP and enables a smooth translation to the VP of those tasks that are not handled efficiently in the MCC without VP (e.g., too much time is required for their handling). With such approach, there is no need to transfer the other tasks to the VP.

For the purpose of program transfer, stand-alone vector systems implement translators of the commonly used programming languages with capabilities of automatic vectorization. However, this same utility is employed to create new programs for the stand-alone vector systems. Such approach is not the only one possible for the VP within the MCC. Under certain conditions, it is economically more advantageous for the MCC with VP to create new programs in languages that have been developed to allow for the specific features of vector processing. Therefore, the main attention in creating the software for the VP was concentrated on the programming system and the language resources for achievement of maximum effective utilization of the VP. The VP is oriented to solving computational problems, and all functions involving organization of computations in the MCC with VP have been delegated to the OS of the Elbrus MCC. The possibility of realizing a full-valued complex is guaranteed by the presence of the system program SUPERVISOR, operating in the VP and interacting with the OS of the MCC by a simple protocol using mail boxes, semaphores, and program interrupts.

The work on the software of the VP was carried out along the following lines.

1. Investigations in the area of program layouts in order to select a layout adequately reflecting the processes of information processing in vector computers of all types (both pipeline and array types) and enabling effective realization in vector computers with programmable configuration of the processing pipeline.

2. Investigations in the area of programming languages for vector computers, including development of constructs of high-level programming language to ensure maximum effective use of vector computers with programmable configuration of the processing pipeline.

3. Development of a technology for creation of translators and verification of same during realization of the VP programming system.

4. Practical realization of a program development complex in the BESM-6 in order to carry out experiments and provide the potential user with a functionally complete means of development and debugging of application programs designed for solution in the VP of the Elbrus MCC.

1. Layout of the Programs, Programming Language, and Translator Design Technology

1.1. The Underlying Program Layout

Almost all existing languages, including graphic means of processing of arrays, as well as programming languages of vector computers, belong to the languages with consecutive operator layout of the programs, i.e., they have a layout similar to the traditional scalar processing languages (ALGOL, FORTRAN, etc.). The hallmark of all the above languages is the definition of the semantics for processing of arrays. In fact, we propose to divide the entire collection of programming languages with means of processing arrays into three groups according to the method of definition of the vector processing semantics (more precisely, the method of definition of the vector assignment semantics) and, thus, to speak of three program layouts underlying the programming languages with array processing means. Apart from this, all three program layouts have an identical consecutive semantics of execution of operators, i.e., the following operator will be executed after completion of execution of the previous operator.

The first group of languages interprets the assignment operator, defined on arrays, as a compact notation for the operator of the cycle. An example of the first group of languages is PL/I.

The second group of languages prescribes evaluating the expression on the right side of the assignment operator, followed by transferring the values of the resulting array of such computation to the array defined by the left side of the assignment operator. This most extensive group of languages includes, in particular, ALGOL-68 and the majority of programming languages of vector computers.

In the third group we shall place languages in which the vector assignment operator is a compact notation for a set of elementary assignment operators (EAOs), defined on the like components of vectors, and the execution of a vector assignment operator is defined as an asynchronous parallel execution of the EAOs which comprise it. Such semantics is generalized in natural manner to an assignment operator of array type (AOAT), when multidimensional arrays are used on the left and right sides of the operator.

The third layout, i.e., a consecutive operator scheme of programs with parallel vector assignment operator, was chosen as the basis for design of the underlying programming language of the VP. This scheme adequately reflects the process of information processing in all kinds of vector computers and can be recommended as the basic scheme for designing the programming languages of vector computers.

1.2. The Underlying Programming Language

The underlying high-level programming language was developed to allow maximum effective utilization of the VP with programmable configuration of the processing pipeline. A survey of the basic constructs of this language is contained in [4]. However, several modifications were made in the final version of the underlying language for description of arrays and subarrays. We therefore present examples of the descriptions of static and dynamic subarrays. Say B is a three-dimensional array. The description of the subarray:

```
/M2/ A[3,2] = B[5*,*,1:8.3-2]
```

then defines a two-dimensional subarray A of the reference array B. In the array B, the fifth slice is taken for the first measurement. In the resulting two-dimensional array, the first and last columns are removed and the resulting array is transposed. The resulting array is the subarray A. Of course, no such displacements of the elements of array B occur. The attributes of array A are simply formulated in such manner as to be accessible to the required subarray. An example of the description of a dynamic subarray C is:

```
/ДМ2/ C[2,1](I) = B[*,*,I].
```

Here, 1 is an associated variable (integer). The subarray C is a transposed two-dimensional one-th slice of the array B in the third measurement.

The underlying language allows effective realization of convolution type operations. For example, the scalar product of vectors A and B with the result placed in scalar X can be written as:

CONVOLUTION (X: = + [A*B]).

For the CONVOLUTION operator, the translator generates a code that will be used to construct the configuration of the functional devices with feedback connections, enabling realization of an accumulation mode.

The underlying language has a standard mechanism of macrosubstitutions, the macrodefinitions being taken from either the macrodefinition partition of the module or the library of macrobased crosses.

1.3. The Technology for Design of Translators

In the design of the VP software, major attention was given to developing a technology of translator creation enabling substantial reduction in the implementation time. The objective was a very quick construction of a fully operational translator, perhaps with some reduction in the speed of translation, but without impairing the quality of the generated object code. If the translator has built in means of automatic gathering of statistical data to allow a determination of the effectiveness of execution of the resulting program and the effectiveness of the translation algorithms, it may be used in the early stage of design of the computing system.

Indeed, analysis of statistical data assembled during the translation of a rather representative collection of programs makes it possible to identify "bottlenecks" in the architecture. If the bottlenecks are discovered early enough, then the hardware can be modified, the translator modified, and the collection of statistics resumed with the same set of actual programs. Comparison of the results of experiments makes it possible to assess the effectiveness of the changes introduced and select the optimal version of the architecture. Furthermore, the means of gathering statistical data that are built into the translator allow an investigation of the process of translation and an optimization of the translation algorithms in the same set of actual programs.

Such approach is effective even in the design of computers with traditional architecture and conventional system of instructions, since the overall effectiveness of the system is optimized in an early phase. For computers with a set of pipeline devices, with programmable switching, and with "very long instruction word" (VLIW systems), such approach is evidently the only one possible, since for VLIW systems the translator should be designed, if not in advance of the hardware, then at least simultaneous with it.

The following technology for development of translators is proposed to support such approach to the design of computing systems.

1. Create an experimental parametrized translator with built-in means of automatic gathering of statistical data.

2. Gather statistics in application program packages and analyze the results.

3. Feedback to the design of the hardware (possible partial redesign on the basis of the analysis findings).

4. Feedback to creation of the experimental translator (variation of parameters; translator version for the improved hardware; optimization of translation algorithms on the basis of the analysis findings).

5. Create a commercial version of the translator.

In order for this technology to be viable, quick implementation and easy modification of the experimental translator must be secured. This, in turn, requires creation of a set of tools, as well as selection (or creation) of a utility language capable of meeting the above requirements. As the utility language, we selected REFAL/2, implemented in many Soviet computers and used in the creation of translators. The writing of translators in REFAL makes them mobile, so that the technology is independent of the specific development computer. Since REFAL/2 was absent from the Elbrus MCC, in the course of the work on the VP software a realization of REFAL for the Elbrus MCC was implemented [7].

The proposed design technology was tested in the development of the VP. We may note the factors with negative impact on the duration of the design process and realization of the underlying language translator.

1. The start of design of the translator was delayed, since the underlying language was being developed along with the VP.

2. No experience in design of optimizing translators for VLIW systems.

3. No utilities available at the start of the project.

4. No sufficiently representative applied program package in the underlying language of the VP necessitated development of a converter of FORTRAN into the underlying language.

Use of REFAL as the development language made it possible not only to shorten the time of implementation of the translator, but also reduce the translator design time. Under one year was spent on the design and implementation of the translator of the underlying language. The translator generates optimal code for the vector segments of the program and carries out a far-reaching optimization during translation of scalar linear segments.

Analysis of statistical data obtained in the experimental translator made it possible to identify certain bottlenecks in the VP architecture at such design stage when partial redesign was comparatively painless. Changes were made in the hardware of the VP, and the experiments to optimize the translation algorithms were continued in the improved (with modified hardware) translator. Not all of the discovered bottlenecks were corrected, but a compromise was achieved, in which further hardware expenses (for the particular componentry) are commensurate with the anticipated outcome of the improvements. In any case, the hardware designers were fully aware in adopting these compromise decisions.

The experience with the design of the VP confirmed the viability of the proposed technology for design of translators. In addition, there is every reason to expect that the period of implementation of an experimental translator for VLIW systems can be further reduced, since experience is now available and a set of utilities has been created.

Let us note that such technology in no way obviates such reliable design tool as the model investigation. On the contrary, the gathering and analysis of statistics complement the model investigation of the operation of the processor in solving test problems. This method is perhaps less exact than a model investigation, but it produces a good picture (beyond the power of a model investigation) of the presence of bottlenecks in the architecture of the processor (in actual programs) and is much more powerful than modeling in the quantity of programs analyzed.

We emphasize once more that the proposed technology is especially effective in design of VLIW systems.

2. Program Development Utility Set

The VP development system includes the following components:

- the translator of the underlying language;

- the translator of the machine code of the VP;

- a compiler of object modules;

- an interpreter of the instruction system;

- an interactive debugger;

- utility programs.

2.1. The Underlying Language Translator

The underlying language translator in processing the descriptions of arrays and subarrays generates instructions that form the values of the attributes. Only a single register of the index register file (IRF) is assigned by the translator for statically equivalent attributes. For subarrays, instructions are generated to check when the boundary of the reference array is crossed, if such static check is not possible. If the name of the dynamic subarray is found in the partition of operators, a special instruction is generated to check whether the value of the associated variable belongs to the permitted range. Instructions to recompute the address of the beginning of the dynamic subarray are generated only if it is possible to change the value of the associated variable after the last recomputation. For AOAT, in addition to address instructions and partial reconfiguration instructions, the translator generates an instruction to load the length of the vector and an instruction to load the time diagrams of the startup and slowdown phase, on the basis of which the VP hardware accomplishes a dynamic integration of consecutive operators [2].

The single algorithm for translation of AOAT, DOUBLET and FORK constructs includes the following main phases.

1. Syntax check and construction of a representation of the computational grid for the given AOAT.

2. Processing of the resulting representation of the grid with a wave algorithm. The relative moments of time (in clock periods) for all nodes of the grid are determined and the contours (if present) are checked for balance [4].

3. Modification of the grid to ensure the necessary separation of scalars (which may occur on the right side of the AOAT) in different sections of the real register file (RRF) and enable a relative shift of nonconnected subtrees (for the DOUBLET construct) which appear in the grid.

4. Appointment of the physical numbers of the devices.

5. Generation of instructions.

A general case of scalar processing may be presented in the form of computations along a linear sequence of assignment operators of scalar type (AOST), not considering control transfer operators, the influence of which on the efficiency of the computations should be considered separately. The AOST may include both scalar variables and constants located in the RRF and elements of arrays contained in the RAM of the VP.

The underlying language translator performs, for the linear segment of scalar processing, a planning of the sequence of switching of the functional devices, the sections of the RRF, and the memory exchange flows, taking into account the delays of the devices and the following limitations: in each clock period, only one pair of operands may arrive at any given functional device, and each section of the RRF may receive/send only one number. Further limitations arise from the information dependents among different AOSTs, i.e., the need to solve the problems of read after write (RAW), write after read (WAR), and write after write (WAW).

The translation of each AOST is done as follows. First, a representation of the tree of computations is constructed, corresponding to the right side of the AOST. A correspondence is established between the operations (internal nodes of the tree) and the functional devices which realize them. The index expressions are translated into a sequence of instructions, which load the working registers of the IRF with the addresses of the required elements of the arrays. A syntax check is made and the addresses of the variables and constants appearing in the AOST are taken from the table of names of the translator.

Next, a recursive circuit of the resulting tree is made and the sequence of switching of the VP devices to accomplish the computations efficiently for the given tree is determined. While moving from the crown to the root of the tree, the possibility of a no-conflict (with respect to computing resources) integration of two previously processed subtree-operands is analyzed for the next node when the operands are simultaneously presented at the inputs of the particular functional device. For the subtrees that have been processed, time diagrams of the occupied resources have already been constructed in the form of a linear sequence of binary scales (one scale corresponds to one clock period of the VP). The position code in the scale represents each of the available devices that is busy during the particular clock period. Comparing the time diagrams period by period makes it possible to analyze the possibility of a combination of subtrees.

If there is a conflict in resources, an attempt is first made to load the lower subtree for four clock periods (the amount of the delay to write the intermediate result in the RRF and read it immediately into the configuration). If the loading for four clock periods is prevented by resource conflict, the lower subtree is moved away from the higher one, first by four clock periods and then by another clock period each time until the conflict disappears. The intermediate result corresponding to the root of the higher subtree will be entered into the RRF register and then read from it. In certain cases, a delay line is used to save the intermediate result. If the next node of the tree corresponds to a unary operation, no sorting through alternatives is required.

For an individual AOST, the translator constructs the following structures:

- a list of appearances of the scalars used, in which the minimum and maximum reading time are specified for each scalar used on the right side of the AOST;

- a time-ordered list of appearance of index variables, giving for each appearance the number of the flow, the address for the IRF register in which the address for the RAM of the VP is formed, and the time of access to the flow (number of clock period);

- a list of partial reconfiguration instructions for the processed tree of computations;

- the height of the processed tree in clock periods;

- the time diagram of occupation of devices in the above-described form.

Translation of the sequence of AOSTs is done in recursive manner in two stages: translation of the next AOST and maximum possible "loading" of the program for the current AOST into the program performing the computation of the previous AOSTs. Maximum loading is prevented both by possible resource conflicts and by information dependences. In general, for a language with high degree of parametrization, especially the underlying language of the VP, the memory addresses for the elements of the array are statically unknown, which enforces a rigorous observance of the sequence of exchanges with memory between the various AOSTs.

For the processed portion of the linear sequence of AOSTs, in addition to the structures indicated for the AOSTs (in the present case referring to the entire processed portion of the sequence of operators) the translator has a list specifying the appearances of the scalar variables, in which the address in the RRF and the maximum writing time at this address are given for each scalar used on the left side of the operator.

After processing the current AOST, the translator loads it in the already processed sequence of AOSTs as far down as permitted by the possible information relationships. If the time diagrams conflict in resources, the current AOST is "floated up" period by period until the conflict vanishes. The algorithm then merges the described structures and moves on to process the next operator.

After processing the entire sequence of AOSTs, the instructions are generated. The translator forms the sequence of instructions of the linear segment of the AOST in the same way as a vector macrobased cross, and the hardware of the VP accomplishes the maximum possible dynamic integration of consecutive macrobased crosses [2].

2.2. The Compiler of Object Modules

The object modules obtained as a result of translation from the underlying language or the machine code of the VP are further processed by the compiler. On the basis of information contained in the object modules, the compiler assembles several separately translated modules into a single module. The modules needed for the compiler to work are taken from the archive of object modules. The result of the work is placed in the archive under a title given by the user. The names of the head module and the result module are given to the compiler as parameters. In addition, the user may specify a filter, indicating the names of modules inaccessible to the compiler. In this case, the indicated modules will not be included in the resulting compilation.

The process of compilation is broken up into two phases. In the first phase, the junction interface between program modules is checked. For this, a table of names of the called compiler modules is constructed, bringing together all names of modules appearing in the compilation with an indication of the relative address of the particular module in the result module, and associated information as to the structure of the parameters of the modules appearing in the compilation. If incorrect module calls are found, the compiler sends diagnostic messages about this. At the end of the first phase, the compiler composes a list of names of modules appearing in the compilation and their relative loading addresses and sends it to be printed.

Transition to the second phase of compilation occurs only if all program module calls are correct. In the second phase, the resulting object module is composed. The call instructions are finally composed in the code field and the other fields of the object module are composed. The resulting compilation contains all information as to the modules appearing therein needed for debugging.

If all called object modules are present in the archive and available to the compiler, the outcome of the full compilation will be a self-contained program complex that can be sent on for execution. But if certain modules are absent from the archive or not available to the compiler, a partly compiled module will be created, containing the object codes of the accessible modules, while the names of the unavailable modules will be entered into a corresponding table of the resulting module.

The compiler has a broad array of diagnostic messages, allowing the user to obtain objective information as to the course of the compilation, as well as certain data on the resulting module obtained.

2.3. Debugging Features

The development system of the VP includes debugging features in the form of an interpreter of the instruction system and an interactive debugger in the terms of the underlying language. These are intended for debugging, investigation of the structure and course of program execution for the purpose of subsequent optimization.

The interpreter of the instruction system simulates the VP hardware and imitates the computation of the user program in the VP by the resources of the development computer. The main requirements on the development of the interpreter were: minimization of the program execution time for the VP, computational results consistent with the actual ones, and a means of program debugging and profiling.

The interpreter is built as a collection of subprograms which simulate the operation of the following devices of the VP:

- the instruction buffer;

- the index processor with IRF;

- the exchange devices between the RAM of the VP and the RAM of the MCC;

- six input and two output flows;

- six functional devices and four sections of RRF;

- the configurator.

The operating algorithms for the modules of the functional devices reflect the operation of the actual equipment. The modules of the devices making up a configuration work with the appropriate time delays. For the divider, this is 16 clock periods, for the adders and multipliers it is 8 periods, for the RRF 3 periods. The interpreter simulates only the configuration element, while the other devices of the VP are interpreted. The VP RAM and the MCC RAM are simulated on external magnetic media in the actual volumes.

The interpreter works closely with the interactive debugger. This is designed to debug user programs and system programs of the VP. The debugging is done in interactive mode in the terms of the underlying programming language of the VP.

Only self-contained (i.e., obtained by full compilation) program modules belong to the debugger. When running programs in the interpreter, the debugger searches and retrieves modules in the system archive.

The debugger makes it possible to establish and remove check points by name of label or by number of macrobased cross (simple, multiple, multiple with a set limit); to initialize the simulation of the program being debugged; to examine the values of the variables (in decimal or octal form) and their attributes; to examine and change the values of certain registers of the VP; to obtain information about the modules making up the user archive; and to keep a record of the debugging.

The possibility is provided to gather statistical information on the programs as they are interpreted. The program profiling features built into the interpreter gather information on the run time of the program and its individual parts with the purpose of an optimization, and also obtain other dynamic statistics regarding the user program. The debugger processes this information and puts it out to a terminal. An important feature of the execution of programs in the interpreter is the possibility of a debugging one clock period at a time. As the program is executed, diagnostic and debugging information appears on the screen; the user may block the appearance of this information by an appropriate adjustment. The speed of the interpreter when processing instructions of the index processor is 1000 clock periods per second, when processing configuration instructions it is 400 periods per second.

3. Results of Experiments

Performance of experiments in the development system of the VP was an integral part of the design process of the VP, having the objective of discovering bottlenecks in the VP architecture, optimization of the translation algorithms, and determination of the efficiency of using the VP in vector and in scalar processing.

The possibility of maximum efficiency of the VP during vector processing was established at a relatively early stage of the project [5]. However, it was only possible to establish (with high degree of reliability) the effectiveness of the VP in scalar processing during the course of an experiment in the development system, making use of a sufficiently representative application program package. For this reason, the main attention in the experiments was concentrated on scalar processing.

In order to assess the effectiveness of scalar processing, the translator of the underlying language has built-in features for gathering of statistics on a number of criteria, enabling a determination of the average speed (clock periods per operation), the relative frequency of possible sources of conflict in resources during attempts to establish a static parallel processing of a problem, the relative frequency of occurrence of a conflict in different situations, the average displacements when joining two subtree-operands together and when joining the next AOST to the previous ones. The parameters gathered by the translator are as follows:

- the total number of jointures of subtree-leaves;

- the total number of nonconflict jointures of leaves;

- the total number of jointures of nontrivial subtrees;

- the number of nonconflict jointures of nontrivial subtrees;

- the number of unsuccessful attempts to join nontrivial subtrees with the smaller subtree loaded for four clock periods;

- the total of resultant upward displacements of the smaller subtree relative to the larger one when joining nontrivial subtrees;

- the total of resulting displacements of an AOST with respect to previous ones;

- the total of minimum boundaries dictated by the information dependence of displacements of the AOST relative to the previous ones;

- the total number of jointures of an AOST to the previous ones;

- the total number of operations in all AOSTs;

- the overall time (in clock periods) for the groups of partial reconfiguration instructions corresponding to all AOSTs;

- the total number of resource conflicts recorded (both in processing the AOST and in joining the AOST to the previous ones) individually for each resource (incoming flows, outgoing flows, functional devices, sections of the RRF).

Statistics were gathered in two groups of tests.

The first small group of seven tests (565 scalar floating-point operations) contained programs for evaluation of an elementary function, programs for solving differential equations, and several fragments of user programs. This group of tests was characterized by being replete with scalar computations of real-valued type.

In order to gather statistics on a more representative class of problems, a limited converter from FORTRAN to the underlying programming language of the VP was developed, adequately converting the scalar processing segments.

The second group of 96 tests (3322 floating-point operations) consisted of subroutines converted from FORTRAN, included in the LIBRARY 1, 2 of standard programs of the Dubna monitor system of the BESM-6. Programs were taken from different partitions of these libraries for the testing:

- polynomials and special functions;

- operations on functions and solving of differential and integral equations;

- interpolation and approximation;

- solving of systems of linear equations;

- nuclear physics and chemistry;

- problems of optimization, model investigation, and control methods;

- high energy physics.

The conversion was done (as is possible in certain cases) without vectorization and involved only the real-value processing segments, including simple transfer (assignment). It was observed that such transfers in the programs of the second group occupy a considerable portion of the real-value processing, which is the reason for the lower speed (in clock periods per operation) for the second group of tests. Furthermore, the programs of the second group were written for the user computer, and they abound in branch instructions, so that the average length of continuous (linear) segments is not large.

We present the averaged statistical parameters (giving in brackets the results for the less-representative first group of tests).

1. The percentage of nonconflict combinations of subtree-operands in an AOST without relative displacement is 90.7 percent (90.5 percent).

2. The percentage of nonconflict combinations of subtrees that are not terminal nodes in an AOST without displacement is 66.2 percent (70.2 percent).

3. The percentage of nonconflict combinations of a subtree with a terminal node in an AOST without displacement is 93.4 percent (96.3 percent).

4. The average displacement of the smaller subtree relative to the larger one in an AOST is 0.37 (0.01) clock periods (loading for four clock periods is taken with negative sign).

5. The average minimum shifting of the current AOST relative to the preceding ones as a result of information dependency is 14.3 (43.4) clock periods.

6. The average shifting of an AOST relative to the preceding ones with allowance for resolution of resource conflicts and information dependencies is 14.7 (48.8) clock periods.

7. Statistics on the causes of resource conflicts:

- memory exchange flows 10.8 percent (31.8 percent);

- divider - 0.5 percent (1.8 percent);

- RRF sections - 78.9 percent (43.2 percent);

- logic device - 2.9 percent (0.1 percent);

- adders S0, S1 - 2.3 percent (0.6 percent);

- multipliers U0, U1 - 4.5 percent (22.5 percent);

8. Average speed is 8.8 (3.8) clock periods per operation.

For the individual AOST, the best indicator of the speed in the tests is 2.2 clock periods per operation (corresponding to a speed of 9 MFLOPS). For a sequence of AOSTs, the best speed in specially selected tests was 0.73 clock periods per operation (29 MFLOPS).

On the basis of the obtained statistical data, the following conclusions can be drawn with respect to the efficiency of the VP in scalar computations.

1. The most critical resource is the RRF sections.

2. The high percentage of causes of data flow conflicts for the first group of tests is not characteristic of the general case and is explained by the presence in this group of a test that is a long linear segment, comprising operators of the form:

    <SCALAR> : = <EXPRESSION PROCESSING ELEMENTS OF ARRAYS>.

For such segment between different AOSTs, there are no information dependences if the scalars on the left side are different, and in the absence of constraints the planning algorithm performs a considerable number of searches among alternatives. The speed for this test (143 operations) amounted to 1.1 clock periods per operation.

3. The most serious impediment in combination of the execution of different AOSTs making up a linear segment is the information dependency.

4. The speed of the VP in scalar processing is roughly an order of magnitude lower than the speed in vector processing.

As a result of the experiments and analysis of statistics in realistic programs, the translator was modified to improve the quality of the code generated. Thus, the workload of the devices S0, S1, U0, U1 became more evenly balanced as a result of the modifications, and the information dependencies were resolved more accurately. The need for further modification of the scalar processing translation algorithm to achieve a more evenly balanced workload of the RRF sections was identified.

Conclusion

Let us list the main results obtained in the development of the software of the VP of the Elbrus MCC.

1. A sequential operator scheme of programs with parallel vector assignment operator was proposed as the conceptual foundation for the design of programming languages for vector computers.

2. On the basis of the proposed layout of programs, a high-level underlying programming language was developed for the VP of the Elbrus MCC, containing sophisticated means of description of subarrays and allowing maximum efficient utilization of the VP with programmable configuration of the processing pipeline.

3. A technology has been developed for construction of translators that is especially efficient in the design of pipeline computing systems with programmable configuration of the processing pipeline and with very long instruction word (VLIW systems). This technology has been tested out in the design of the VP.

4. As part of the project, a version of the language REFAL/2 for the Elbrus MCC was created.

5. A VP development system has been realized in the BESM-6, operating under control of the OS DISPAK, in the presence of the ARFA archive system. In addition, translators of the underlying language and the VP machine code have been implemented in the Elbrus MCC.

6. Experiments conducted in the VP development system have produced a qualitative understanding of the bottlenecks in the architecture of the VP and made it possible to establish the effectiveness of using the VP in both vector and scalar processing. Furthermore, the experiments enabled an optimization of the translation algorithms both with respect to translation time and quality of generated code.

Bibliography

1. Burtsev, V. S., et al., "A Vector Processor for the Elbrus MCC Family," "Aktualnyye problemy razvitiya arkhitektury i programmnogo obespecheniya EVM i vychislitelnykh sistem" [Current Issues in Development of the Architecture and Software of Computers and Computing Systems], Computer Center, Siberian Division, USS Academy of Sciences, Novosibirsk, 1983, pp. 3-12.

2. Burtsev, V. S., et al., "A Vector Processor with Programmable Structure," VYCHISLITELNYYE PROTSESSY I SISTEMY, Issue 2, Nauka, M., 1985, pp. 63-72.

3. Asriyeli, V. D., Borisov, P. V., "A Processor with Programmable Switching," Preprint 9, Institute of Precision Mechanics and Computer Technology im. Lebedev, USSR Academy of Sciences, M., 1982, 17 pp.

4. Asriyeli, V. D., "The Underlying Language of the Vector Processor Programming System," VYCHISLITELNYYE PROTSESSY I SISTEMY, Issue 2, Nauka, M., 1985, pp. 73-83.

5. Asriyeli, V. D., Borisov, P. V., "Experience in Programming the Solving of the Navier-Stokes Equations in a Three-Dimensional Region for the Vector Processor," VYCHISLITELNYYE PROTSESSY I SISTEMY, Issue 2, Nauka, M., 1985, pp. 84-90.

6. Myasnikov, A. V., "A Program Development Complex for the Elbrus MCC Vector Processor," "Tez. dokl. konferentsii molodykh matematikov" [Abstracts of Reports of a Conference of Young Mathematicians, Sverdlovsk, January 1987], Sverdlovsk, 1987, pp. 62-63, (Preprint, Institute of Mathematics and Mechanics, USSR Academy of Sciences, Urals Division).

7. Popov, T. A., "The Programming System REFAL/2 for the Elbrus MCC," "Tez. dokl. konferentsii molodykh matematikov" [Abstracts of Reports of a Conference of Young Mathematicians, Sverdlovsk, January 1987], Sverdlovsk, 1987, pp. 64-65, (Preprint, Institute of Mathematics and Mechanics, USSR Academy of Sciences, Urals Division).

ARCHITECTURE AND ORGANIZATION OF THE DIAGNOSTIC UTILITY SYSTEM OF THE VECTOR PROCESSOR OF THE E-2 MCC

907G0089D Moscow SUPER-EVM in Russian, pp. 67-76

[Article by N. S. Fetisov, M. V. Tverdokhlebov, A. A. Sablukov, and A. N. Kramarenko]

[Text] The expanded functional capabilities and, thus, more complicated hardware of computers, computing complexes and systems, and the increased demands on the operating reliability of the systems, give rise to the necessity of developing methods of technical diagnostics. Such methods are involved in many areas of the engineering sciences in data processing and computer technology. Issues of inspection, diagnostics, and reliability are considered during the development of the architectures of computing systems and the structural principles of implementation of the computer devices, in the circuit engineering, and in the development of the computer software.

The complicated choice of a method of inspection and diagnosis of a particular system results both from the broad spectrum of existing methods of implementation of such task and from the contradictory demands on the system of inspection and diagnosis of each particular system under development. A further degree of complexity in the handling of such task is brought in by the necessity of optimization of the solution, taking into account the high cost of the diagnostics in the hardware and other expenses and, especially, the impossibility of achieving certain methods of diagnostics on account of the existing technical constraints and demands for a particular product. As a result, an integrated approach is usually employed to handle the area of inspection and diagnostics of the hardware, combining elements of architectural, structural, circuit-engineering and software reliability and technical diagnostics.

One of the ways of handling the question of detection and finding of faults in digital equipment is the use of the method of a shadow shift register, implemented (in particular) in several projects of the IBM company, as well as in the development of the vector processor (VP) of the Elbrus-2 multiprocessor computing complex (E-2 MCC) [1]. In the VP of the E-2 MCC, this method is implemented in a somewhat modified form.

The essence of the shift register method involves the combining of all functional flip flop elements of a digital device into a single shift register, granting a hardware possibility of reading and writing in this register by means of a shift operation. For this, it is possible either to use microcircuits with an auxiliary shift function as the functional shift registers or to employ a duplication of the functional flip flop elements with copying of information into "shadow" flip flop elements not used in normal functioning of the equipment.

Implementation of such possibility affords read and write access to any given flip flop element within a digital device (having a minimum number of external connections) and, in providing a hardware interface, makes it possible to utilize the obtained information in handling the task of inspection and diagnosis in a certain processing device, external with respect to the equipment under diagnosis.

The external processing device (terminal computer) for the VP of the E-2 MCC is the minicomputer SM-1420 [2], provided with a special interface to the VP and special software. The assemblage of these devices is known as the diagnostic utility system of the VP of the E-2 MCC [3]. The authors have worked out the architecture and organized the software of this system. The system is intended for use in the debugging phase, and also for inspection and diagnostics during operation of the VP.

Development of the ideology and hardware implementation of the shift register method in the equipment of the VP was accomplished by the development engineers of the vector processor of the E-2 MCC [1, 4]. Powerful hardware support of the tasks handled by the diagnostic utility system is accomplished, in particular, by a special built-in utility device of the VP, which besides providing communications with the SM-1420 terminal computer supports all kinds of interaction between the terminal computer and the VP [4].

The connection of the diagnostic utility system (DUS) to the VP is shown schematically in Fig. 1. Although the DUS was specially developed for the VP, it can also be used for any other synchronous digital equipment in which a hardware interface is provided to the DUS, and the flip flop elements in the circuits are combined into a single (possibly shadow) shift register.

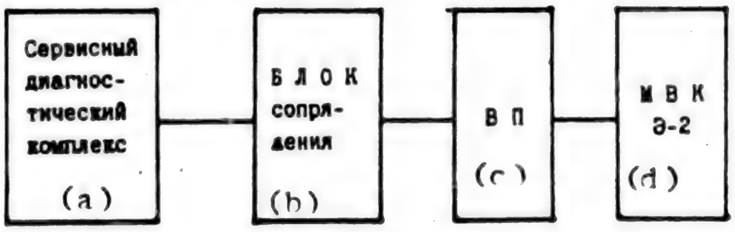

Fig. 1. Connection of the DUS to the VP.

Key:

a.	Diagnostic utility system	c.	VP
b.	Interface unit	d.	E-2 MCC

The structure of the diagnostic utility system is shown in Fig. 2. The firmware interface to the VP enables a read and write exchange by a snapshot or shift chain between the VP and the working storage of the SM-1420 terminal computer, loading in the VP and starting of the functional inspection test of the VP, stopping of the VP, changing of the operating modes of the VP, starting the VP, and also performance of other functions stipulated by the instruction system of the interface unit.

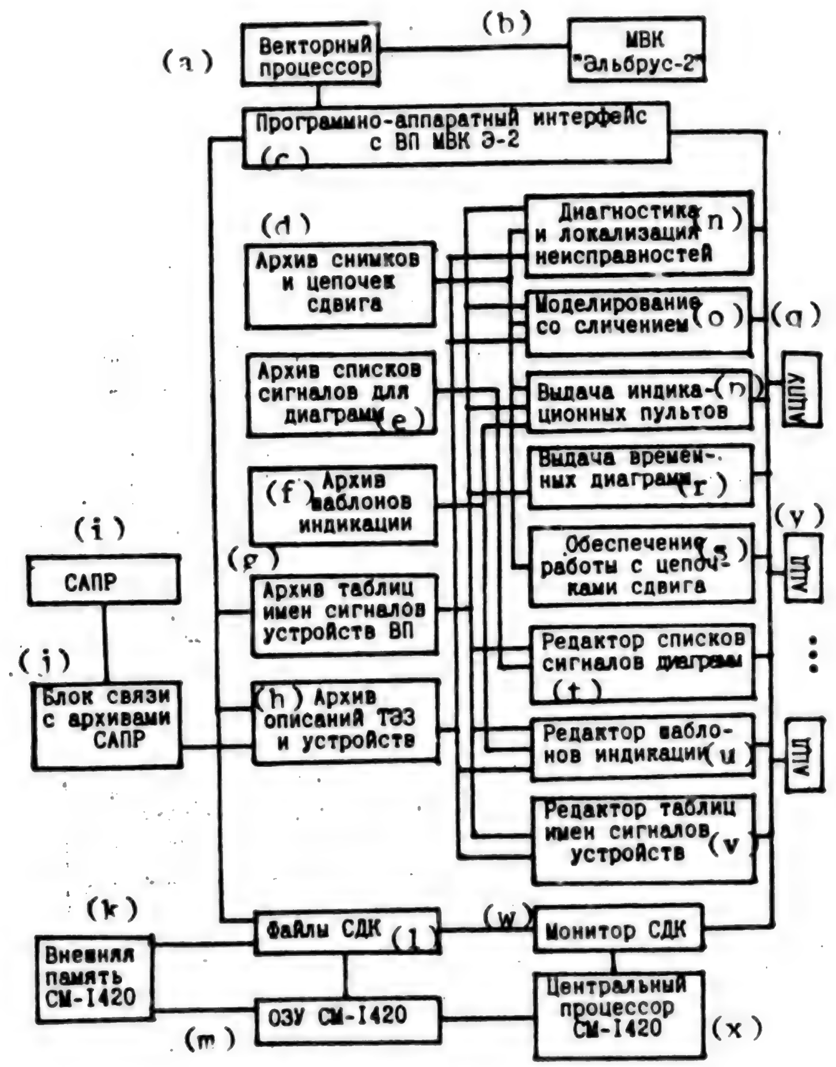
(a) Векторный процессор
(b)
МВК "Эльбрус-2"
Программно-аппаратный интерфейс с ВП МВК Э-2 (c)
(d) Архив снимков и цепочек сдвига
Архив списков сигналов для диаграмм (e)
(f) Архив шаблонов индикации
(g) Архив таблиц имен сигналов устройств ВП
(h) Архив описаний ТЭЗ и устройств
(i) САПР
(j) Блок связи с архивами САПР
(k) Внешняя память СМ-1420
Файлы СДК (l)
(m) ОЗУ СМ-1420
(w) Монитор СДК
Центральный процессор СМ-1420 (x)
Диагностика и локализация неисправностей (n)
Моделирование со сличением (o)
(q) АЦПУ
Выдача индикационных пультов (p)
Выдача временных диаграмм (r)
Обеспечение работы с цепочками сдвига (s)
(y) АЦД
Редактор списков сигналов диаграмм (t)
Редактор шаблонов индикации (u)
АЦД
Редактор таблиц имен сигналов устройств (v)

Fig. 2. Structure of the diagnostic utility system.

Key:

a.	Vector processor
b.	Elbrus-2 MCC
c.	Firmware interface to the VP of the E-2 MCC
d.	Archive of shift chains and snapshots
e.	Archive of lists of signals for diagrams
f.	Archive of indication patterns
g.	Archive of tables of names of signals of VP devices
h.	Archive of descriptions of cars and devices
i.	CAD
j.	Link to CAD archives
k.	SM-1420 external memory
l.	DUS files
m.	SM-1420 RAM
n.	Diagnostics and localization of faults
o.	Modeling with verification
p.	Output of display consoles
q.	Alphanumeric printer
r.	Output of time diagrams
s.	Support of operation with shift chains
t.	Editor of lists of signals of diagrams
u.	Editor of indication patterns
v.	Editor of tables of names of signals of devices
w.	DUS monitor
x.	SM-1420 central processor
y.	Alphanumeric display

As noted, all flip flop elements are combined into shift registers or chains in the shift register method. The totality of all chains for a device or the processor as a whole is known as a snapshot. It is necessary to point out in the following discussion that the number of flip flop elements (stages) in the chain is unknown. The sequence of chains in the snapshot is also unknown.

Thus, the DUS is able to control the operation of the VP and obtain information from the VP in the form of snapshots. On the other hand, the DUS is provided with a link to the archives of a computer-aided design (CAD) system, thus obtaining access to the archive descriptions of the devices of the VP, consisting of a list of the elements and a list of the interconnections and names of the signals, the names of the cards and modules. All of these attributes are represented in the schematic diagrams of the product and constitute the basis for an understanding of the product, its operation and repair.

Since this information is (of course) absent from the electronic circuits and, thus, from the snapshots obtained by the DUS via the link to the VP, the DUS should be able to perform a one-to-one transformation of the numbers of the flip flops from the shift chain into the terms of the circuit documentation and vice versa.

In order to handle this task, the DUS is provided with tables of names of the signals for each device of the VP, which is kept in the archive of tables of correspondence of the names of the signals of the VP devices. The table establishes a correspondence between the name of the signal in the diagram of the card of a particular device and the number of the bit in the shift chain.

Using the name of the signal and the name of the card for a particular device, the table makes it possible to find the number of the bit in the corresponding shift chain and, on the contrary (as is easily seen in Fig. 3), the name of the signal and the name of the card to which this signal belongs is found from the number of the bit of the shift chain of a particular device. The organization of the tables of correspondence of the signal names (CSN) allows identical signal names to be present in different cards. On the other hand, replacement of the signal names and card names with address references (Fig. 3) enables a substantial reduction in volume of memory required to store these tables, which is of definite value when the DUS is implemented in a minicomputer.

The entry of the CSN tables was resolved in the DUS by developing a special entry language and editor of the tables of correspondence of signal names of the devices, making extensive use of indexing of the register signals and the numbers of the stages of the shift chain when entered into the table by the user. The indexing makes it possible to reduce the volume of work of user entry of the CSN table by several times (tens of times for register signals).

The mechanism of the CSN tables thus allows a transformation of the user directives, expressed in terms of circuit documentation, into the language of the DUS-VP interface, i.e., in terms of the numbers of stages of the shift chains.

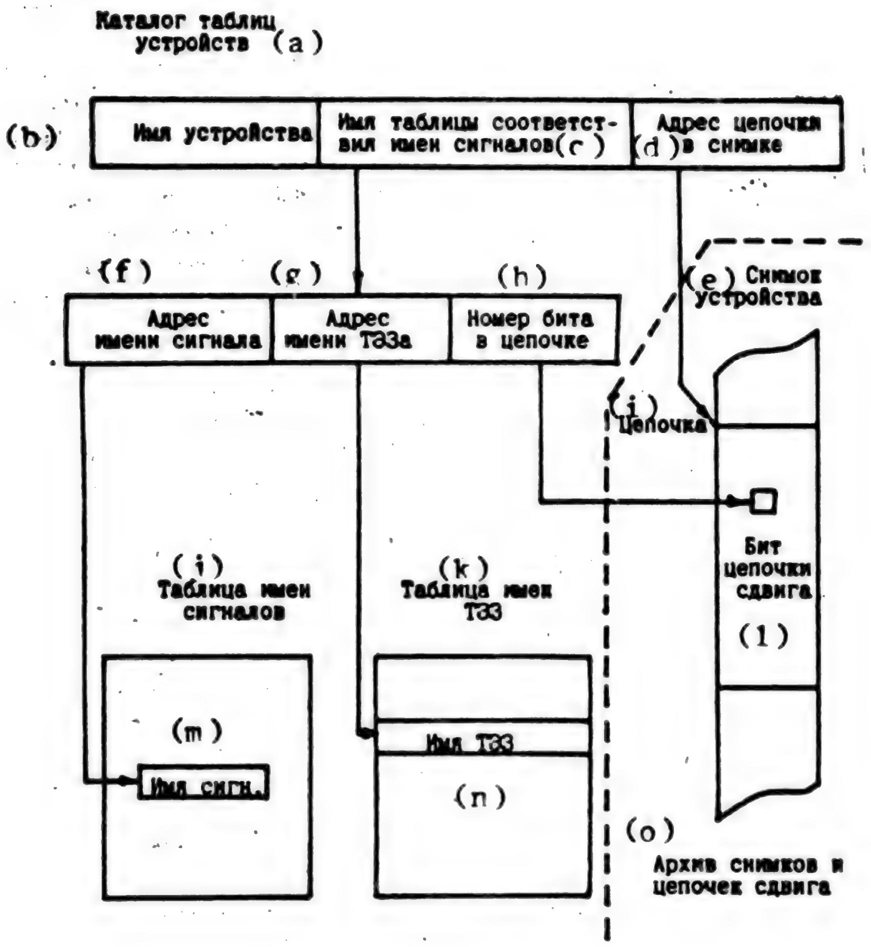

Fig. 3. Organization of the operation of the table of correspondence of signal names.
Key:

a.	Catalog of tables of devices	h.	Number of bit in chain
b.	Name of device	i.	Chain
c.	Name of table of correspondence of signal names	j.	Table of signal names
d.	Address of chain in snapshot	k.	Table of card names
e.	Snapshot of device	l.	Bit of shift chain
f.	Address of signal name	m.	Signal name
g.	Address of card name	n.	Card name
		o.	Archive of snapshots and shift chains

The main tasks handled by the diagnostic utility system from the standpoint of the user of the DUS are as follows:

- indication of the static condition of the flip flop elements of the VP;

- dynamic indication from one clock period to another with output of the VP operation time diagrams;

- modeling of individual circuits and devices of the VP from one clock period to another on the basis of an archive with comparison against the VP operating diagrams from one clock period to another;

- diagnostics and localization of VP faults.

In the article, we shall discuss only the first three of these tasks. The method of diagnostics and localization of faults by use of shift registers has no direct bearing on the subject of the present work, which is mainly devoted to the architecture and organization of the DUS. We merely note that granting of access to all flip flop elements of the digital circuit substantially facilitates the solving of the problem of diagnostics and synthesis of test sequences. In such approach, it is possible to use the most simple and well-studied methods of analysis and synthesis of tests for combination layouts. However, in practice, the problem is somewhat more complicated, since not all flip flop elements have been incorporated into a shift chain in the VP under consideration. Furthermore, the situation is greatly complicated by the need to analyze not only hard failures, but also soft failures in the functioning of the equipment, especially in the case when the hardware monitoring sends the emergency stop signal much later than the appearance of the error in operation of the equipment.

Indication of the static condition of the flip flop elements of the VP involves:

- formation, editing, and output to the terminal and printer of indication patterns;

- output of the values of the shift chain or snapshot of the VP by the indication patterns, the names of the registers, and the names of the individual signals.

The indication pattern here is a certain product of the display console, the format of which will be used subsequently to put out an indication from the VP. In the DUS, the screen of the alphanumeric display of the terminal computer is used as the display screen for the VP. Any desired number of indication patterns may be prepared in advance and stored in the indication pattern archive of the DUS. A pattern is the contents of one terminal screen, filled in by the indication pattern editor in a special mode. In the places where it is necessary to show the value of a particular register or individual signal during the indication, the title of such is directly entered. During the indication, the title is replaced with the value from the shift chain or snapshot. Any desired alphanumeric labels or notes may be placed in the field of the screen. These remain unchanged during indication and make it possible to create legible displays. The values of the registers can be put out in base 2, 8, or 16 arithmetic systems, and also in the form of an integer and a real value.

Dynamic indication from one clock period to another with output of the operating time diagrams of the VP is secured by the organization of a special mode of operation of the VP with storage of the sequence of snapshots of the VP or chains of a specified device in the DUS archive during each clock period, as well as a program block for management and editing of the signal lists for the time diagrams. Different versions of such lists may be written into the archive of signal lists for the diagrams.

After a given sequence of snapshots of the operation of the VP is written into the archive of snapshots and shift chains, this sequence may be used on many occasions to investigate any given diagrams by any given user of the DUS over a lengthy period of time. This ensures a minimum time of occupation of the VP, which is important in the case of debugging or repairing of the VP, allowing the VP to be used from several DUS terminals at the same time.

The sequences of snapshots formed in the DUS archive are also used for comparison against the results of the modeling of individual circuits and devices of the VP during each clock period on the basis of their archive description. Adequate functioning of the model and equipment depends much on the choice of an appropriate method of modeling, avoiding possible discrepancies between the operation of the model and that of the equipment. The issue of adequacy of modeling methods is discussed, in particular, by [5].

In addition to the above, the functions of the DUS include:

- placing the values of the shift chain in symbolic names and editing thereof;

- comparing a list (chain) obtained from hardware against a snapshot (chain) previously stored in the archive of snapshots of the DUS; such option allows a diagnosis of the VP in terms of stored sequences of snapshots during short tests; in event of discrepancy, the names of the flip flops and the numbers of the clock periods at the time of the discrepancy are put out;

- checking the switching of the indicated group of flip flops in a given test for preliminary analysis of completeness of the test.

The database of the diagnostic utility system enables storage of all types of information used with explicit indication of the type and program checking for its proper use. The information is protected against unauthorized access.

The software of the DUS was developed in the framework of the operating system RAFOS SM-1420 (a counterpart of the system RT-11 of the DEC company) [6], although in the standard mode the DUS runs under control of its own monitor, specifically oriented to carry out the group of DUS programs. In addition to the main requirement of control of the DUS, the monitor was developed to ensure:

- minimization of the memory volume of the resident portion of the monitor;

- substantially increased reliability of information storage, including protection against soft failures during the operation of the file system (as compared to the OS RAFOS);

- effective operation with information arrays in the working storage, the size of which is greater than 64 K byte, thanks to direct use of additional registers.

The last point is especially important in assuring sufficient speed of modeling of devices of the necessary extent.

In the information structure of its archives, the DUS monitor is fully compatible with the OS RAFOS. Interesting in their own right are the screen-line text editor with dynamic segmentation of the file being edited and protection against soft failures, as well as a debugger program for operation with assembler programs, which were developed in the context of the software of the DUS. These programs can be used by themselves within the OS RAFOS.

The DUS monitor supports simultaneous operation of as many as four terminals, and also enables execution of the DUS operators in interactive and batch modes.

Bibliography

1. Burtsev, V. S., Krivosheyev, Ye. A., et al., "A Vector Processor with Programmable Structure," in VYCHISLITELNYYE PROTSESSY I SISTEMY, Issue 2, edited by G. I. Marchuk, Nauka, M., 1985.

2. Vigdorchik, G. V., Vorobyev, A. Yu., Prachenko, B. D., "Osnovy programmirovaniya na assemblere SM EVM" [Fundamentals of Programming in the Assembler of the SM Computer], FINANSY I STATISTIKA, M., 1983.

3. Fetisov, N. S., Tverdokhlebov, M. V., "Organization and Software of the Diagnostic Utility System," "Tezisy dokladov konf. MS i chlenov NTORES im. A. S. Popova," [Abstracts of Reports of the Conference of MS and Members of NTORES im. Popov], Section 1, Computer Structure, ITM and VT AN SSSR, M., 1987.

4. Koreshkov, V. D., "A Built-In Utility for the Vector Processor," "Tezisy dokladov konf. MS i chlenov NTORES im. A. S. Popova," [Abstracts of Reports of the Conference of MS and Members of NTORES im. Popov], Section 1, Computer Structure, ITM and VT AN SSSR, M., 1987.

5. Fetisov, S. N., "Nekotoryye voprosy modelirovaniya i analiza testov tsifrovykh ustroystv," [Several Aspects of Modeling and Analysis of Tests for Digital Devices], ITM and VT AN SSSR, M., 1980.

6. Valikova, L. I., Vigdorchik, G. V., Vorobyev, A. Yu., Lukin, A. A., "Operatsionnaya sistema SM EVM RAFOS: Spravochnik" [The Operating System SM EVM RAFOS: Manual], ed. by V P. Semik, FINANSY I STATISTIKA, M., 1984.

SEVERAL STRUCTURAL AND ALGORITHMIC ASPECTS OF A SCALAR-VECTOR COMPUTATIONAL MODEL

907G0089E Moscow SUPER-EVM in Russian, pp. 77-96

[Article by A. S. Olenin]

[Text] Specialization is one of the most important ways of increasing the speed of computations. From the standpoint of data processing, computations are divided into two major groups: numerical and logical, so that any given algorithm may be characterized by a number $I = Q_l/Q_n$, where Q_n and Q_l are the number of numerical and logic operations, respectively, in it. This number serves as a criterion of the "intellectuality" of the algorithm and characterizes to some degree its area of computation. Thus, when $I \ll 1$, the computations should be characterized as substantially numerical, and when $I \gg 1$ as substantially logical. The concepts for design of the computational equipment for each of these regions may differ significantly from each other. In the first case, for example, we require processors oriented to numerical solution of scientific and engineering problems, and in the second case processors which handle information in logical space.

The early stages of development of computer equipment involved the creation of universal computers capable of covering a relatively small region of I values on either side of unity and therefore meeting the practical needs for processing of numerical and logical information only in limited extent. The present stage of computations requires ever further penetration into the regions of small and large numbers I, and we are witnessing how special devices or processors meeting these needs are being created in the context of universal machines or in self-standing form.

The domain of extremely low I is of major importance to the numerical solution of scientific and engineering problems which demand a steady increase in computational work. The increase in volume of computations should not entail a major growth in the volume of the programs. Hence the desire to structure the data in such way that their dimension does not control the length of the program notation. This approach corresponds to a vector or matrix organization of data, which in many cases requires a special formulation of the algorithm. Actual algorithms generally contain both vector and scalar (i.e., nonvectorizable or poorly vectorizable) sections and a certain amount of combining logic. The scalar component is characterized by the now customary and traditional computations in the region of I numbers much closer to unity than the initial algorithm as a whole, and therefore these can be realized effectively by universal means. Thus, one of the ways of entering the domain of supernumerical computations $I \ll 1$ is breaking up the algorithm into scalar and vector components and applying a universal structure to the first of these, and a specialized vector processor based on pipeline, matrix, or some other organization to the second. The concept of a program in its fundamental sense is retained, being supplemented primarily by the introduction of linguistic vector constructs.

The process of computation may be represented by a sequence of steps of type:

$$q = a\theta b,$$

where a, b, q are the vectors of the operands, θ is the vector of operation signs +, -, x, etc.

We may distinguish several cases of interpretation of this expression, comparing them in simplified fashion with the types of computations.

1. Scalar computations may be defined by a sequence of steps:

$$q_0 = a_0 \vartheta b_0,$$

which differs from the above in that the operands and the sign of the operation in each step are scalars.

2. Vector computations involve a representation of the algorithm by a collection of steps of type:

$$q = a \vartheta b,$$

where the operands are vectors, while the signs of the operations are scalars.

3. Parallel computations directly correspond to the notation:

$$q = a\theta b.$$

Each of these expressions can be matched up with its own type of computational structure in the familiar classification: the first with SISD, the second with SIMD, the third with MIMD. Hereafter we shall be concerned only with the first two types of computation, assuming that if the vectors of the signs of the operations in the latter expression contain a large number of identical elements then it is advisable to disjoin the parallel computations into more simple blocks of vector type, having separate control for their own scalars of the operations.

The most simple model of scalar-vector computations may be described, e.g., by the sequence of steps:

$$\tau = f(a', b').$$
$$q = a \vartheta \tau b,$$

where f(a', b') is a scalar function of vectors which specifies the scalar parameter for the following vector process. This model admits an analogy with the operational portion of the axiomatics of a vector space over a field of real numbers, granting the possibility of an exchange of information between the space and the field. This immediately delineates the contours of a scalar-vector calculator which assumes the presence of two interacting processors, one of which processes the field, the other the space. We shall call such structural formation a scalar-vector complex (SVC).

If during each step of the model it is possible to combine in time the performance of a vector operation and the calculation of a scalar parameter to be used in the next vector operation, we shall call such process a scalar-vector process with disjunction. The practical value of a scalar-vector model with disjunction is that the scalar component in this case may be performed in the background of the vector computation, in no way hindering it. In conditions of disjunction, the speed of the computation depends on the speed of the vector processor, and the complex problems of accelerating the scalar component when it becomes a "bottleneck" may be simply unimportant. Of course, this requires that the vector component surpass the scalar so greatly in volume of computations that even the use of a vector processor would not result in a computational time advantage for the scalar component.

In the actual situation, scalar-vector computations should be understood in a much broader sense than that in the elementary model. This is especially true of the function of a scalar processor (SP), which is entrusted with the support of the operation of the vector processor (VP), in addition to the immediate computations. The disjunction of an algorithm into scalar and vector parts should be done for reasons of effectiveness and balance in the operation of the complex. The scalar component may keep the unutilized options of vectorization, which will be implemented in event of overloading of the vector processor.

If the range of user problems contains a vector component of sizable volume and if the scalar computation is not the "bottleneck" of the computations, the speed of the complex (as already noted) will depend on the speed of the VP, and the highest priority of the scalar processor should be to ensure an uninterrupted, fully loaded operation of the vector component. The operating time of the SP not devoted to this function may be used in multiprogram mode for other less important tasks. Such mode enables high speed both in an individual task and in a mixture of tasks: in the first case, thanks to the VP, in the second, thanks to the SP.

The design principles for the Soviet complex for scalar-vector computations were advanced by V. S. Burtsev in the course of the development of the Elbrus-2 MCC [1] and the vector processor [2]. Implementation of these principles required a solution to a number of special hardware and system problems. At the same time, in the context of the topic of mutual representation of computing structures and algorithms, studies in the organization of algorithms to ensure effective functioning of SVC are becoming important. Summarizing the above, the following are the main requirements on the properties of the algorithms:

1. A vector quality. Vectorization of the algorithms should strive to make the volume of vector calculations in them much greater than the volume of the scalar part.

2. Loading properties of the algorithms. The vector component should ensure (where possible) full workload of the functional devices of the VP in the course of its processing. A full workload should be taken to be a processing

of vectors with constant and not overly small size in sufficiently large computational segments.

3. Disjunction of computations into scalar and vector components. By disjunction (as we have seen) is understood not merely a breakup into scalar and vector components, but a construction (if possible) of such combination thereof that the scalar element can be executed in the background of the vector computation, in no way hindering it.

The special nature of the above qualities involves the fact that their realization occurs on the algorithmic level, regardless of the particular structure of the SP and VP. We present below a brief discussion of several algorithms for solving systems of linear algebraic equations (SLAE) by using the aforesaid properties in an application to a model of the SVC. Discussion of the detailed estimates, comparisons, and applications to specific architectures exceeds the ambition of the present work. General approaches to the parallel running of a number of special algorithms of mathematical physics will be found, e.g., in [4, 5].

1. Scalar-Vector Algorithmics of Dense Matrices

We shall assume that the matrix A in the system $Ax = f$ is dense, real-valued, nondegenerate, and we consider a model of SVC that realizes the following construction in each "n"-th step of the computations:

$$\tau_{n+1} = f(a'_n, b'_n), \tag{1}$$

$$q_{n+1} = a_n - \tau_n b_n, \tag{2}$$

where a_n, b_n are vectors, τ_n is a scalar function of vectors. As the discussion proceeds, the model will be supplied with the necessary supplements where necessary.

The scheme (1), (2) is a scalar-vector process with disjunction on the basis of the vector process (2). The vector process (2) will be termed complete if the dimensions of the vectors being processed are kept unchanged from one step to another. In this case, a complete workload of the functional devices of the VP can be automatically assured and there is no need to monitor the length of the vectors during the computation. We define the scalar τ_n as the parameter of the vector process (2) in the "n"-th step. Processes with vector parameters that will be encountered further on call for an element by element vector operation of multiplication, instead of a scalar one.

Let us give some characteristic examples. Say it is required to evaluate the expression:

$$\xi = f - Ax,$$

where x, ξ, f are vectors, A is a matrix of order N with columns a_n. This expression may be written naturally in the form of a vector process:

$$s_n = s_{n-1} - x_n a_n, \quad s_0 = f, \quad \xi = s_N, \quad n = \overline{1,N}.$$

The role of the parameters of the process is played here by the components of the vector x. If the matrix A is dense, the vector process is complete. But in the case of a triangular matrix, for example, the completeness of the process is lost, since in order to avoid operations on the zero elements it is necessary to change the length of the vector from one step to another and to deal with short vectors at a particular stage. This, of course, entails further expenses on organization of the computations and has a negative impact on the operating efficiency of the processor. Vector processes that are formed on the basis of triangular matrices, i.e., such as have a linear change in the length of the vector from one step to another, will be termed triangular. We note that the method of doubling, e.g., for the summation of numbers, results in a logarithmic change in the length of the vector as a function of the number of steps and can be written as a logarithmic vector process.

From the standpoint of complete workload of the VP, complete vector processes with length of vector not excessively short have the most efficient mapping onto the structure of the vector processor, and therefore we shall devote the main attention to them hereafter.

It should be noted that the construction of complete vector processes may involve not only a lowering of the height of the algorithm, but also a certain increase in its length--the total number of operations. Therefore, the advisability of using complete algorithms depends on how much more advantageous in terms of computational time are their more pronounced vector properties than the short length of their less vectorial counterparts. With increasing parallel processing resources, the vectorial qualities of complete algorithms become predominant, and when the resources are sufficiently large their use is the most efficient.

1.1. Direct Methods

The Jordan algorithm is naturally written as a complete vector process ($a_{N+1} = f$, $x = a_{N+1}{}^{(N)}$):

$$\tau_j^{(k)} = \frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}}, \tag{3}$$

$$a_j^{(k)} = \mathring{a}_j^{(k-1)} - \tau_j^{(k)} \mathring{a}_k^{(k-1)}, \quad k = \overline{1,N}, \quad j = \overline{k+1,N+1}, \tag{4}$$

where

$$a_j^{(k-1)} = \left(a_{1j}^{(k-1)}, \dots, a_{k-1,j}^{(k-1)}, a_{kj}^{(k-1)}, a_{k+1,j}^{(k-1)}, \dots, a_{Nj}^{(k-1)}\right)^T,$$

$$\mathring{a}_k^{(k-1)} = \left(a_{1k}^{(k-1)}, \dots, a_{k-1,k}^{(k-1)}, 1, a_{k+1,k}^{(k-1)}, \dots, a_{Nk}^{(k-1)}\right)^T,$$

$$\mathring{a}_j^{(k-1)} = \left(a_{1j}^{(k-1)}, \dots, a_{k-1,j}^{(k-1)}, 0, a_{k+1,j}^{(k-1)}, \dots, a_{Nj}^{(k-1)}\right)^T.$$

The method of Gauss elimination is also characterized by a vector form, but the calculation now involves two stages in succession. The forward path occurs by a scheme analogous to (3), (4), where the vectors are defined somewhat differently:

$$\tau_j^{(k)} = \frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}}, \tag{5}$$

$$\bar{a}_j^{(k)} = \dot{\bar{a}}_j^{(k-1)} - \tau_j^{(k)} \dot{\bar{a}}_k^{(k-1)}; \quad k = \overline{1,n}, \; j = \overline{k+1, n+1}, \tag{6}$$

and

$$\bar{a}_j^{(k-1)} = (0, 0, \ldots, 0, a_{kj}^{(k-1)}, a_{k+1,j}^{(k-1)}, \ldots, a_{nj}^{(k-1)})^T,$$

$$\dot{\bar{a}}_k^{(k-1)} = (0, 0, \ldots, 0, 1, a_{k+1,k}^{(k-1)}, \ldots, a_{nk}^{(k-1)})^T,$$

$$\dot{\bar{a}}_j^{(k-1)} = (0, 0, \ldots, 0, 0, a_{k+1,j}^{(k-1)}, \ldots, a_{nj}^{(k-1)})^T.$$

The bar here and afterwards indicates abbreviated vectors, which are formed from the complete ones, as is obvious, by replacing the portion of the elements above or below the diagonal with zeros. The dot above a vector indicates that the k-th row of the matrix $A^{(k-1)}$ in the k-th step is replaced by another row (in the present case, the k-th row of the unit matrix).

After (5), (6) is executed, reverse substitution is accomplished by a triangular vector process. Processes (5), (6) are also triangular in the index k.

The Jordan vector scheme as compared to the Gauss has lower height and less completeness of the process, but is ~1.5 times greater in length. Therefore, from the standpoint of traditional consecutive computations, the method of elimination may be regarded as a certain optimization of the Jordan scheme for the case of poorly parallel structures, in particular, a single processor structure, when the time of execution of the algorithm depends primarily on its length. The decrease in length by switching to the Gauss method is achieved at the cost of a certain loss in vector properties of the Jordan scheme by partial scalarization, forming two consecutively executed segments: a forward path and a backward path. Similar methods of optimization are the method of optimal elimination, the compact scheme of the Gauss method, and other techniques that are effective in the sense of a consecutive realization of the SLAE solution algorithm.

The decomposition of a dense matrix into a Frobenius matrix product results in a complete vector process:

$$A = \prod_{i=1}^{n} \Phi_i, \qquad x = A^{-1}f = \prod_{i=n}^{1} \Phi_i^{-1} f,$$

where

$$\Phi_i = \begin{bmatrix} 0 & 1 & \dots & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & 1 \\ \delta_{i1} & \cdot & \cdot & \delta_{in} \end{bmatrix}, \quad \Phi_i^{-1} = \begin{bmatrix} -\frac{\delta_{i2}}{\delta_{i1}} & \dots & -\frac{\delta_{in}}{\delta_{i1}} & \frac{1}{\delta_{i1}} \\ 1 & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & 1 & 0 \end{bmatrix}.$$

Applying this decomposition to the transposed system $x^T A^T = f^T$, we may see that it corresponds to the Jordan scheme. It is easily observed that the Jordan scheme (3), (4) is a disjoining scalar-vector process, since the parameters $r_j^{(k)}$ may be computed for different j in the background of execution of the vector process.

We may observe that introducing a procedure of searching for the leading elements into algorithms of the elimination type disturbs the regularity of the computation and, thus, reduces the effectiveness of the vector computations. From this standpoint, and for reasons of precision, the methods based on orthogonal transformations are preferable, even though they require a larger number of operations. Let us consider the vector qualities of two such methods: the method of reflections and the method of rotations.

In the method of reflections, the initial system is transformed into a triangular one by means of an orthogonal reflection matrix [3]:

$$U = E - \frac{2ww^T}{\|w\|^2}.$$

If the vector w is chosen by formulas $w = a - \sigma e$, $\sigma = \sqrt{(a, a)}$, then the given vector a will become a collinear unit vector e by means of the transformation $Ua = \sigma e$. As applied to the solution of SLAE, the method is written thus:

$$\sigma_k = \sqrt{(\bar{a}_k^{(k-1)}, \bar{a}_k^{(k-1)})}, \quad w_k = \bar{a}_k^{(k-1)} - \sigma_k e_k,$$

$$\bar{a}_j^{(k)} = \bar{a}_j^{(k-1)} - \frac{2(w_k, \bar{a}_j^{(k-1)})}{(w_k, w_k)} w_k, \quad k = \overline{1,n}, \; j = \overline{k+1, n+1}.$$

Designating:

$$w_{kj} = a_{kj} - \frac{(\bar{a}_k^{(k-1)}, \bar{a}_k^{(k-1)})}{\sigma_k},$$

we have

$$\bar{a}_j^{(k)} = \bar{a}_j^{(k-1)} - r_j^{(k)} w_k$$

where we obtain for the parameter:

$$\tau_j^{(k)} = \frac{2(w_k, \bar{a}_j^{(k-1)})}{(w_k, w_k)} = \frac{(\bar{a}_k^{(k-1)}, \bar{a}_j^{(k-1)}) - \sigma_k a_{kj}^{(k-1)}}{-w_{kk}\sigma_k} = \frac{w_{kj}}{w_{kk}} .$$

Finally, we arrive at the following algorithm, transforming the original system into a triangular one:

$$w_{kk} = a_{kk}^{(k-1)} - \sigma_k, \quad \sigma_k = \sqrt{(\bar{a}_k^{(k-1)}, \bar{a}_k^{(k-1)})},$$

$$w_{kj} = a_{kj}^{(k-1)} - \frac{(\bar{a}_k^{(k-1)}, \bar{a}_j^{(k-1)})}{\sigma_k}, \quad \tau_j^{(k)} = \frac{w_{kj}}{w_{kk}},$$

$$\bar{a}_j^{(k)} = \dot{\bar{a}}_j^{(k-1)} - \tau_j^{(k)} \dot{\bar{a}}_k^{(k-1)}, \quad k = \overline{1,n}, \quad j = \overline{k+1, n+1}.$$

As we see, the only difference between this scheme and the Gauss scheme is the method of computation of the parameters. The process is triangular and the height of its vector component evidently coincides with the Gauss scheme. In order to convert to a complete process with Jordan height, we convert the abbreviated vectors to complete ones or combine the computation of the portion of the matrix below the diagonal with a Jordan elimination in the triangular system. In the first case, the parameter is one and the same over the entire length of the vector, while in the second case there are two quantities acting as the parameters, being computed in terms of the scalar products of the abbreviated vectors.

The method of rotations is accomplished by means of elementary rotation matrices. Say we have a matrix $A^{(k-1)}$, in which the elements below the diagonal in columns 1 through k-1 are equal to zero. We introduce the notation:

$$\sigma_{ij}^{(k-1)} = \sum_{p=k}^{i} a_{pk}^{(k-1)} a_{pj}^{(k-1)} = (\bar{a}_k^{(k-1)}, \bar{a}_j^{(k-1)})_i, \quad i = \overline{k,n}.$$

The matrix $A^{(k)} = T^{(k)} A^{(k-1)} = T_{nk} T_{n-1,k} \dots T_{k+1,k} A^{(k-1)}$

will contain zero elements below the diagonal in the k-th column as well, provided that:

$$T_{ik} = \begin{bmatrix} 1 & \cdot & \cdot & 0 \\ \cdot & c_{ik} & -s_{ik} & \cdot \\ \cdot & s_{ik} & c_{ik} & \cdot \\ 0 & \cdot & \cdot & 1 \end{bmatrix}, \quad i = \overline{k+1, n}.$$

and the elements of the matrices T_{ik} are found by formulas:

$$c_{ik} = \sqrt{\frac{\sigma_{i-1,k}^{(k-1)}}{\sigma_{ik}^{(k-1)}}}, \quad s_{ik} = -\frac{a_{ik}^{(k-1)}}{\sqrt{\sigma_{ik}^{(k-1)}}}.$$

Performing the multiplication of the elementary matrices belonging to the k-th column, it is easily seen that, using the equality

$$\sigma_{ij}^{(k-1)} = \sigma_{i-1,j}^{(k-1)} + a_{ik}^{(k-1)} a_{ij}^{(k-1)}$$

the conversion of the elements during the elimination is done by the formula:

$$a_{ij}^{(k)} = \frac{1}{c_{ik}} \left(a_{ij}^{(k-1)} - \frac{\sigma_{ij}^{(k-1)}}{\sigma_{ik}^{(k-1)}} a_{ik}^{(k-1)} \right).$$

Omitting the coefficient in front of the parentheses, on which the result of the solving of the system does not depend, but the orthogonality of the transformation does, we arrive at the expression:

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{\sigma_{ij}^{(k-1)}}{\sigma_{ik}^{(k-1)}} a_{ik}^{(k-1)}.$$

Since $c_{ik} \leq 1$, this expression may correspond to lower values of the intermediate elements than the previous one. As a result, the following triangular process is created:

$$\tau_j^{(k)} = \frac{\sigma_{ij}^{(k-1)}}{\sigma_{ik}^{(k-1)}},$$

$$\bar{a}_j^{(k)} = \dot{\bar{a}}_j^{(k-1)} - \tau_j^{(k)} \times \dot{\bar{a}}_k^{(k-1)}, \quad k = \overline{1,N}, \; j = \overline{k+1, N+1}.$$

We may arrive at a complete vector process by replacing the abbreviated vectors with complete ones or (as above) by combining this scheme with a Jordan elimination for a triangular system. Unlike the schemes given above, the parameter here is a vector.

We present a general scalar-vector elimination scheme for solution of SLAE. For this, we write the system Ax = f in the form:

$$\sum_{j=1}^{N} x_j a_j = f.$$

We shall eliminate the unknowns by a scalar multiplication of the left and right sides with vectors that are orthogonal to the respective columns. The vector z is defined by means of relations:

$$z = s - ct, \quad c = \frac{(s, a_k)}{(t, a_k)},$$

where s, t are arbitrarily specified nonzero vectors, c is a constant chosen from the condition $(z, a_k) = 0$. If the unknown x_k is to be eliminated from $M(\underline{M < N})$ equations, we must accordingly construct the set of vectors z_i, $i = \overline{1, M}$, satisfying conditions $(z_i, a_k) = 0$. Hereafter, it is assumed that $s_i = e_i$, and thus we have:

$$z_i = e_i - \frac{a_{ik}}{(t, a_k)} t,$$

$$(z_i, a_j) = a_{ij} - \frac{(t, a_j)}{(t, a_k)} a_{ik}.$$

Using this formula to solve a SLAE produces the following complete scalar-vector algorithm:

$$\tau_j^{(k)} = \frac{(\bar{t}_k, \bar{a}_j^{(k-1)})}{(\bar{t}_k, \bar{a}_k^{(k-1)})}, \tag{7}$$

$$a_j^{(k)} = \dot{a}_j^{(k-1)} - \tau_j^{(k)} \dot{a}_k^{(k-1)}, \quad k = \overline{1, N}, \quad j = \overline{k+1, N+1}, \tag{8}$$
$$x = a_{N+1}^{(N)}.$$

A similar scheme occurs for transformation of the original system into a triangular one:

$$\tau_j^{(k)} = \frac{(\bar{t}_k, \bar{a}_j^{(k-1)})}{(\bar{t}_k, \bar{a}_k^{(k-1)})}, \tag{9}$$

$$\bar{a}_j^{(k)} = \dot{\bar{a}}_j^{(k-1)} - \tau_j^{(k)} \dot{\bar{a}}_k^{(k-1)}, \quad k = \overline{1, N}, \quad j = \overline{k+1, N+1}. \tag{10}$$

Let us consider several cases of specification of the vector $\bar{t}_k$.

1. $\bar{t}_k = e_k$. The parameters in this case are given by relations:

$$\tau_j^{(k)} = \frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}},$$

i.e., algorithm (7), (8) passes into the Jordan method, while algorithm (9), (10) into the forward path of the Gauss method.

2. $\bar{t}_k = k_k^{(i)} = (0, \ldots, 0, 1, 0, \ldots, 0)^T$, where the ones are located at positions k through i. Computation of the parameters comes down to the following ratio:

$$\tau_j^{(k)} = \sum_{p=k}^{i} a_{pj}^{(k-1)} \Big/ \sum_{p=k}^{i} a_{pk}^{(k-1)}, \qquad i \geq k$$

3. $\bar{t}_k = \bar{w}_k = (0, \ldots, 0, w_{kk}, a_{k+1\,k}^{(k-1)}, \ldots, a_{Nk}^{(k-1)})^T$

where

$$w_{kk} = a_{kk}^{(k-1)} - \sigma_k, \qquad \sigma_k = \sqrt{(\bar{a}_k^{(k-1)}, \bar{a}_k^{(k-1)})}.$$

In this case we have:

$$\tau_j^{(k)} = \frac{(\bar{w}_k, \bar{a}_j^{(k-1)})}{(\bar{w}_k, \bar{a}_k^{(k-1)})} = \frac{(\bar{a}_k^{(k-1)}, \bar{a}_j^{(k-1)}) - \sigma_k a_{kj}^{(k-1)}}{-\sigma_k w_{kk}} = \frac{w_{kj}}{w_{kk}},$$

$$w_{kj} = a_{kj}^{(k-1)} - \frac{(\bar{a}_k^{(k-1)}, \bar{a}_j^{(k-1)})}{\sigma_k}$$

and an algorithm with such parameter is equivalent to the above-considered scheme for the method of reflections.

4. $$\bar{t}_k^{(i)} = (0, \ldots, 0, a_{kk}^{(k-1)}, \ldots, a_{ik}^{(k-1)}, 0, \ldots, 0)^T, \quad i > k.$$

We obtain an expression for the parameter:

$$\tau_j^{(k)} = \frac{(\bar{a}_k^{(k-1)}, \bar{a}_j^{(k-1)})_i}{(\bar{a}_k^{(k-1)}, \bar{a}_k^{(k-1)})_i}.$$

If the index i in this runs through values from k to N in each vector step, the parameters will become vectors and the algorithm will concur with the scheme obtained above for the method of reflections.

5. $\bar{t}_k = \bar{a}_k^{(k-1)}$

Here we have a vector process using all elements of the abbreviated elimination column:

$$\tau_j^{(k)} = \frac{(\bar{a}_k^{(k-1)}, \bar{a}_j^{(k-1)})}{(\bar{a}_k^{(k-1)}, \bar{a}_k^{(k-1)})},$$

$$\bar{a}_j^{(k)} = \bar{a}_j^{(k-1)} - \tau_j^{(k)} \bar{a}_k^{(k-1)}, \qquad k = \overline{1, N}, \quad j = \overline{k+1, N+1}.$$

We note that a scalar-vector notation is also natural for the method of orthogonalization. For example, the column version of this method can be written as:

$$\tau_j^{(k)} = \frac{(a_k, v_j^{(k-1)})}{(a_k, v_k^{(k-1)})},$$

$$v_j^{(k)} = v_j^{(k-1)} - \tau_j^{(k)} v_k^{(k-1)}, \quad k = \overline{1,n}, \; j = \overline{k+1, n+1}$$

in the forward path, and:

$$x_k = \frac{(f_k, v_k^{(k-1)})}{(a_k, v_k^{(k-1)})}, \quad k = \overline{n,1},$$

$$f_{k-1} = f_k - x_k a_k, \quad f_n = f.$$

in the backward path. As we see, both stages represent a complete scalar-vector process.

1.2. Iteration Methods

While realizing that the multitude of iteration algorithms is vast [6], we shall touch on a few of them in order to explain the characteristic features of the scalar-vector model. We shall employ the following canonical form [7] of the iteration methods:

$$B \frac{x^{(n)} - x^{(n-1)}}{\tau_n} + A x^{(n-1)} = f.$$

The matrix B and the iteration parameters τ_n are selected from conditions of best type of convergence and minimum total volume of computations. The specific quality of a given method depends on the specified structure of the matrix B and the method of computing the parameters.

In the explicit case B = E, this expression may be rewritten as two relations:

$$x^{(n)} = x^{(n-1)} + \tau_n r^{(n-1)},$$
$$r^{(n)} = f - A x^{(n)},$$

from which it is obvious that the explicit iteration scheme is naturally represented by a pair of complete vector processes. The set of parameters for this may be found from a priori information as to the spectral properties of the matrix A. In methods of variation type and in schemes of conjugated directions formulated in vector form, the parameters are computed by using current information obtained during the iteration.

The addition of an implicit quality in the method of iterations makes it possible to shorten the number of iterations as compared to the explicit scheme, although in certain cases the volume of computations in each iteration is increased. A triangular matrix or a product of triangular matrices is often used as the matrix B. From the standpoint of vector properties, the change to triangular matrices (as already mentioned) results in insufficiently effective triangular processes. However, by special organization of the computations, the possibility exists of constructing complete vector processes in this case as well.

Let us consider several triangular methods. We shall assume that the matrix A has been reduced to the form A = L + E + U, where L, U are the lower and upper triangular matrices, respectively, with zero major diagonals, and E is a unit matrix.

Consider the iteration scheme:

$$(E + \tau L)\frac{x^{(n)} - x^{(n-1)}}{\tau} + Ax^{(n-1)} = f, \qquad (11)$$

which during each iteration requires inversion of a triangular matrix. The Gauss-Seidel method is obtained when $\tau = 1$. In this case, we have the expression:

$$x^{(n)} = f - Ux^{(n-1)} - Lx^{(n)},$$

which we shall write as two relations:

$$s^{(n)} = f - Ux^{(n)},$$

$$x^{(n)} = s^{(n-1)} - Lx^{(n)}$$

We now consider the following vector procedure for calculation of the n-th iteration:

$$y_1^{(1)} = y_1^{(0)} - a_{11} y_1^{(0)}, \quad y_1^{(2)} = y_1^{(1)} - a_{12} y_2^{(1)}, \ \dots, \ y_1^{(N)} = y_1^{(N-1)} - a_{1N} y_N^{(N-1)};$$

$$y_2^{(1)} = y_2^{(0)} - a_{21} y_1^{(0)}, \quad y_2^{(2)} = y_2^{(1)} - a_{22} y_2^{(1)}, \ \dots, \ y_2^{(N)} = y_2^{(N-1)} - a_{2N} y_N^{(N-1)},$$

$$\dots\dots\dots\dots\dots\dots$$

$$y_N^{(1)} = y_N^{(0)} - a_{N1} y_1^{(0)}, \quad y_N^{(2)} = y_N^{(1)} - a_{N2} y_2^{(1)}, \ \dots, \ y_N^{(N)} = y_N^{(N-1)} - a_{NN} y_N^{(N-1)}.$$

It is easy to verify that this is equivalent to the previous relations by assuming that:

$$y_k^{(k-1)} = x_k^{(n)}, \quad y_k^{(N)} = s_k^{(n)}, \quad y_1^{(0)} = x_1^{(n)} = s_1^{(n-1)}$$

and setting $y_k^{(k-1)} = f_k$, $a_{kk} = 0$ in the k-th row prior to evaluating the vector $y^{(k)}$.

Thus, a complete scalar-vector process for the Gauss-Seidel method will be as follows:

$$x_k^{(n)} = y_k^{(k-1)},$$

$$y^{(k)} = \dot{y}^{(k-1)} - x_k^{(n)} \dot{a}_k, \qquad k = \overline{1,N}, \tag{12}$$

where

$$y^{(0)} = s^{(n-1)}, \quad y^{(N)} = s^{(n)};$$

$$\dot{a}_k = (a_{1k}, \ldots, a_{k-1,k}, 0, a_{k+1,k}, \ldots, a_{Nk})^T;$$

$$\dot{y}^{(k-1)} = (y_1^{(k-1)}, \ldots, y_{k-1}^{(k-1)}, f_k, y_{k+1}^{(k-1)}, \ldots, y_N^{(k-1)})^T.$$

The parameters here are the components of the unknowns, obtained as the computations proceed. The described construction leaves the length of the algorithm almost unchanged, since the complete process is formed by combination of two triangular matrices into a single complete one. Triangular computations remain only in the calculation of the initial approximation.

The method of consecutive upper relaxation corresponds directly to scheme (11), where τ is the parameter of relaxation. Hence, we have the expression:

$$x^{(n)} = (1-\tau)x^{(n-1)} + \tau(f - Ux^{(n-1)} - Lx^{(n)})$$

or

$$s^{(n)} = f - Ux^{(n)},$$

$$\tilde{x}^{(n)} = s^{(n-1)} - Lx^{(n)},$$

$$x^{(n)} = (1-\tau)x^{(n-1)} + \tau\tilde{x}^{(n)}.$$

By analogy with the algorithm (12), the latter relations are equivalent to a complete vector process:

$$x_k^{(n)} = (1-\tau)\, x_k^{(n-1)} + \tau y_k^{(k-1)},$$

$$y^{(k)} = \dot{y}^{(k-1)} - x_k^{(n)}\dot{a}_k, \qquad k = \overline{1,N}, \tag{13}$$

where (as above):

$$y^{(0)} = s^{(n-1)}, \quad y^{(N)} = s^{(n)}.$$

We note that the difference between algorithms (12) and (13) is concentrated in the definition of the parameters, while their vector component is identical.

Let us consider the alternately triangular method in the form:

$$(E+U)(E+L)\frac{x^{(n)}-x^{(n-1)}}{\tau}+Ax^{(n-1)}=f.$$

Here, during each iteration it is necessary to invert a triangular matrix twice. We introduce the notation:

$$z^{(n)}=(E+L)x^{(n)}.$$

Using this, we will have:

$$(E+U)\frac{z^{(n)}-z^{(n-1)}}{\tau}+z^{(n-1)}=s^{(n-1)},$$
$$s^{(n)}=f-Ux^{(n)};$$
$$x^{(n)}=z^{(n)}-Lx^{(n)}.$$

The latter two relations (as previously) may be combined into a complete vector process, where

$$y^{(n)}=z^{(n)}, \quad y^{(n)}=s^{(n)},$$

and inversion of a triangular matrix is only necessary to find the vectors $z^{(k)}$. The starting approximation is computed by formulas:

$$s^{(0)}=f-Ux^{(0)}, \quad z^{(0)}=x^{(0)}+Lx^{(0)}$$

and also admits of a combination into a complete process.

2. Features in the Organization of the Computations

As follows from the above analysis, the characteristic feature of schemes represented as complete vector processes is the fact that they consist of two interrelated parts: scalar and vector. The properties of the algorithms are determined by the scalar part, which specifies the method of computing the parameters for the vector processes. But the vector part, which requires the largest volume of computations, remains almost unchanged. Therefore, it is perfectly natural to employ here the model of the SVC as a structurally similar model. The vector process is used especially to accomplish the vector part of the algorithms. The parameters required for its operation are supplied by the scalar processor, the operation of which takes place in the background of the vector computations. It is clear from the formulas for the generalized elimination scheme that the input data for the vector part are the parameter of the process and the corresponding substituted row. If the functioning of the complex is coordinated so that these data are available to the vector processor at the start of each vector step, the time for the computations depends entirely on the speed of the latter.

In the case of orthogonalization, the scalar and the vector parts are comparable in number of operations, and therefore one should consider the prospect of vectorization of the evaluation of the parameters themselves, in order for the use of the described complex to remain efficient. This is also true of other methods containing an excessively laborious determination of parameters.

For reasons of economy, in the elimination schemes it is better to employ not complete, but partial scalar products in calculating the parameters. Therefore, in order to achieve coordination, it is sufficient to compute during each step only as many terms in the scalar products as can be handled by the scalar processor up to the time when the vector processor will require the corresponding parameter. In particular, the number of terms may be set in advance by specifying a certain number of nonzero elements of a smoothing vector $\bar{t}_k$. With such organization of the computations, it is possible for the scalar processor to generate at the outset a series of parameters which differ in the number of terms in the scalar products and to select the one with the minimum modulus. This allows automatic prevention of situations with loss of precision, without resorting to a special search for the leading element. It is important to emphasize that these operations concern only the scalar part of the algorithm, and therefore the operating mode of the complex with dynamic supply of parameters, while retaining high speed, is capable of providing computations of good precision.

Bibliography

1. Burtsev, V. S., "Principles of Design of the Elbrus Multiprocessor Computing Complexes," Preprint 1, ITM and VT AN SSSR, M., 1977, 53 pp.

2. Burtsev, V. S., Krivosheyev, Ye. A., Asriyeli, V. D., Borisov, P. V., Tregubov, K. Ya., "A Vector Processor with Programmable Structure," in VYCHISLITELNYYE PROTSESSY I SISTEMY, Issue 2, Nauka, M., 1985, pp. 63-72.

3. Voyevodin, V. V., "Chislennyye metody algebry" [Numerical Methods of Algebra], Nauka, M., 1966.

4. Lebedev, V. I., Bakhvalov, N. S., Agoshkov, V. I., Baburin, O. V., Knyazev, A. V., ,Shutyayev, V. P., "Parallelnyye algoritmy resheniya nekotorkh statsionarnykh zadach matematicheskoy fiziki" [Parallel Algorithms for Solving Several Stationary Problems of Mathematical Physics], OVM AN SSSR, M., 1984, 142 pp.

5. Marchuk, G. I., Ilin, V. P., "Parallel Computations in Grid Methods of Solving Problems of Mathematical Physics," Preprint 220, VTs SO AN SSSR, Novosibirsk, 1979, 23 pp.

6. Marchuk, G. I., Kuznetsov, Yu. A., "Iteration Methods and Quadratic Functionals," in "Metody vychislitelnoy matematiki" [Methods of Computer Mathematics], Nauka, Novosibirsk, 1975.

7. Samarskiy, A. A., Nikolayev, Ye. S., "Metody resheniya setochnykh uravneniy" [Methods of Solving Grid Equations], Nauka, M., 1978.

8. Burtsev, V. S., Krivosheyev, Ye. A., Asriyeli, V. D., et al., "The Vector Processor of the Elbrus-2 MCC," (in present volume).

UDC 681.3;519.246.8

SPECTRUM ANALYZER BASED ON AN ELEKTRONIKA-70 MICROCOMPUTER

18630180B Novosibirsk AVTOMETRIYA in Russian No 6, Nov-Dec 88 (manuscript received 16 Oct 1987) pp 23-27

[Article by D.N. Zabiyaka, A.A. Predtechenskiy and A.I. Chernykh, Novosibirsk: "Spectrum Analyzer Based on an Elektronika-70 Microcomputer"]

[Text] Introduction

The transformation of time-varying signals into the frequency domain — spectral analysis — is a powerful tool for studying a wide variety of physical entities: electronic devices, communication channels, acoustic and seismic waves, water waves, hydrodynamic turbulence, the vibrations of machines, etc.

In the sonic frequency range (up to 20-40 kHz), the traditional analog methods of spectral analysis have today been completely replaced by digital methods, such as recursive digital filtering and methods which make use of various versions of fast Fourier transforms [FFT]. These methods are completely indispensable in the spectral analysis of infralow frequencies, and they surpass the digital-analog method of temporal compression of a signal in terms of analysis accuracy.

Recording the power spectrum is not always the ultimate goal: subsequent processing and analysis (sometimes extremely complicated) of the resulting spectrograms are frequently necessary. Recursive digital filtering can be used effectively only for a logarithmic frequency scale; this circumstance is a serious burden for subsequent processing, especially in the case of multichannel analysis. We will accordingly restrict the discussion below to devices which operate on the basis of FFT algorithms. We can distinguish four approaches in the implementation of FFT analysis:

1. **The conventional device approach.** Because of the multitude of analysis parameters, the corresponding instruments — spectrum analyzers — are becoming progressively more complex to control, and large and specialized

control panels with tens of different buttons and knobs are required. The latter frequently operate under the control of a separate microprocessor. Apparatus of this type is being produced by many manufacturers: Bruel & Kjaer (Denmark), Hewlett-Packard (US) and Ono Sokki (Japan). A dedicated FFT processor is built into these instruments to increase their speed. It increases the signal band which can be processed in real time to 2-5 kHz. In general, all the processors of these instruments are narrowly specialized and do not allow the user to add new functions. They are thus not very effective in scientific research.

2. **Stand-alone data acquisition systems**. In this approach the signal is first recorded by a data acquisition system, which is a separate unit usually containing a high-quality multichannel analog-to-digital converter (ADC) and a digital memory of 40-100 Kwords, equipped with an MEK-625 interface. In a system of this type the data processing, including the spectral analysis, is carried out on a personal computer, which is also connected to the system. The computational capabilities and display facilities of the personal computer are utilized. Some typical instruments are the Data Laboratories DL-1200 and DL-1208 (Britain), the Solartron 1250 (Britain) and the Hewlett-Packard 3852 system (US). A positive feature of these systems is that they are open to the user.

3. **Instruments based on general-purpose microprocessors**. In this approach, a microcomputer based on a fast general-purpose microprocessor (usually the MC68000 or the 8086) is built directly into the apparatus which contains the data acquisition system and the display devices. An instrument of this sort is controlled by a small keyboard, and the functions of the various keys can usually be synthesized on the monitor screen and can be changed as the apparatus is adjusted to the desired type of operation. As an example we could cite the Analogic DATA-6000 (Data Precision, US), in which the basic operations are performed by software on an MC68000 microprocessor. More elaborate processing can be programmed by the user in a language of the BASIC type with a calculator keyboard. Systems of this type are obviously open to the user, although the maximum frequency of the data which can be processed in real time is rather low, 200-400 Hz.

4. **Measurement and computation complexes**. The tasks involved in the spectral analysis of signals can also be performed by open complexes based on bus-module CAMAC systems under the control of a minicomputer, with appropriate software and display facilities. Such systems offer formidable analysis capabilities but may be unacceptably large. Examples are the SK4-71 (USSR) and the Intertechnique IN-100 (France) analyzers.

In this connection, there is practical interest in implementing signal analyzers based on the widely available open-architecture microcomputers (with a bus-module organization), e.g., those of the Elektronika 60 family. Ideally, no structural changes would be necessary, and there would be a

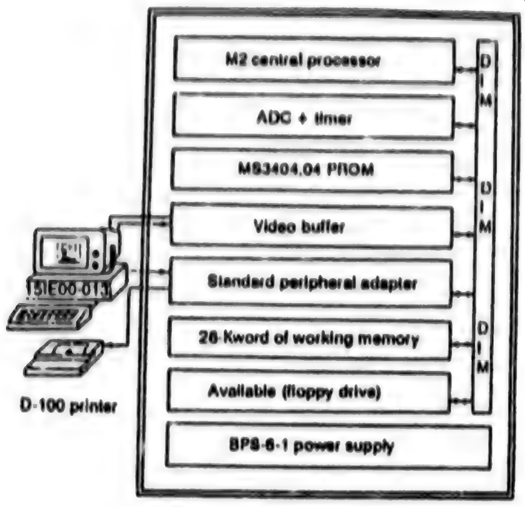

Figure 1.

minimum number of peripheral devices. Our purposes in the present study were to develop the missing hardware components (an input unit and a display controller), to develop appropriate software for processing and control, and to study the capabilities of a single-channel version of the apparatus.

1. Hardware

The architecture of the analyzer has the classic bus-module structure and is shown in fig. 1. The basic assembly of the microcomputer (the central processing unit, the memory and the terminal and printer interfaces) is supplemented with a system for entering analog data and a graphics controller. We should point out that the memory of the computer must be static, so that refreshing by the central processing unit does not interrupt the data stream. The analyzer is designed for stand-alone operation without any peripheral storage devices, so it includes an MS3404.04 programmable read-only memory (PROM) module, which stores the entire program and certain tests. Even if an M2 processing is used in the analyzer, the box of the computer still has room for a half-height board, a serial-communications interface or a floppy drive controller. The capabilities of the analyzer would thereby be expanded in a qualitative way.

1.1. The data input subsystem uses 113PV1 10-digit integrated-circuit ADC; it has two channels and an interface which provides direct access to memory and which is based on a KR580VT57 large-scale integrated circuit. The input of each of the two channels has a symmetric differential circuit in which three KR544UD1A operational amplifiers are used. Each of the operational amplifiers is followed by a KR1100SK2 sampling and storage circuit. The half-height board also contains a programmable quartz clock based on a KR580VI53 large-scale integrated circuit, which sets the data sampling period. The analog section is powered by the ±12-V power supply of the computer through isolating LC filters. There is no galvanic decoupling. Tests have shown that the pickup from the digital devices does not exceed ½ of a digit of the ADC in the input range ±2.5 V. The data acquisition rate can range up to 32-33 kHz, through two channels simultaneously, so a frequency analysis band of up to 16 kHz can be achieved.

The existing version of the data input system does not have a controllable antialiasing filter; the user himself is expected to limit the band of the input signal.

1.2. An important function of the analyzer, if it is to be user-friendly, is to provide a high-quality display of the results of the measurements in graphics form. Unfortunately, graphics terminals are not yet widely available. We attempted to reconcile the standard alphanumeric representation of a 15IE-00-013 display unit with a graphics representation generated by a special video buffer. The display unit itself was not revised. The circuits for sweeping the module of the graphics video buffer are synchronized by the line and frame pulses coming from a cable connecting the video logic of the display unit to its monitor. The data generated by the video buffer are summed with the alphanumeric video signal and sent to the input of the monitor, with the result that a sum of images is formed. The microprogramming unit of the video-memory access ensures a nonflickering image, regardless of software accesses of the video buffer, in any part of the raster sweep cycle. The basic module (on a half-height board) forms an image field of 512 × 256 pixels and has the hardware capability of shifting the entire image at steps of one pixel vertically and eight horizontally. There is provision for increasing the number of memory layers by connecting an additional board.

2. Control Program

As can be seen in fig. 1, the analyzer has no specialized modules for calculating FFT. All of the data processing is carried out by software facilities. The program initializes three pseudoprocessors, which are synchronized in a special way.

The first processor has top priority and controls the data input and the transmission data to the processing processor. The input processor controls the direct-access unit, generating addresses at the appropriate time for the binary data buffering. It starts the data processing, monitoring the processing to see whether it is being carried out in real time. In addition, it synchronizes the triggering of the data acquisition by an input signal: there is a free or slave triggering regime upon a positive or negative front.

The data processing processor is implemented as a pseudomachine which contains a memory of macroinstructions of a special format, which in a sense

Table 1.

Computer (processor)	Execution time, sec	Computer (processor)	Execution time, sec
Elektronika 60M (M2)	1,9	LSI-11/23	0,80
Elektronika 81B	0,85	DVK-2M (MS1201.02)	0,66
Elektronika 60.1 (M6)	0,82	Elektronika 100/25	0,61

Table 2. Comparative Characterisics of Certain Common Signal and Spectrum Analyzers

Model	Manufacturer	Frequency band, kHz	Input: digits (dB)	Channels	Windows	Real time, kHz	Bands	Volume, dm^3	Volume, kg
SK4-71	USSR	0—50	10p	2	?	16	2048	2347	550
2515	«Bruel & Kjaer»	0—20	(60)	1	2	?	1275	24	16
2031	The same	0—20	12p	1	2	2	400	66	22
2033	»	0—20	12p	1	2	2	4000	66	22
2034	»	0—25,6	12p	2	4	1,6	801	66	35
3582A	«Hewlett — Packard»	0—25,5	(70)	1	3	?	256	44	24
3561A	The same	0—100	(80)	1	4	7,5	400	39	22
3562A	»	0—100	(70)	1	6 +	10	800	54	26
CF-200	«Ono Sokki»	0—20	12p	1	2	0,2	400	18	8
CF-920	The same	0—100	12p	2	6	2	800	54	28
Analyzer of present study		0—16	10p	1	3 +	0,3	4096	116	51

constitute a problem-oriented language of the analyzer. A sequential interpretation of the macroinstructions by an execution subprogram arranges the data processing in the necessary order. The processor has a regime of editing of the macroinstructions, so there is some flexibility in the processing. The memory is automatically reallocated under the buffer files with the appropriate control. In a single-channel version, the following set of macroinstructions was realized: multiplying an input-data vector by several standard windows, taking direct and inverse Fourier transforms, forming spectral amplitudes, linear and exponential averaging, taking the logarithm of the spectrum, scaling the spectrum and generating a graphics representation.

The control processor synthesizes on a graphics-free field of the screen a multilevel menu; it processes the user's instructions. A particular line on the menu is chosen by pressing the key with the appropriate number once; the effect is to maximize the speed at which the apparatus is controlled. In the specification of numerical values, extensive use is made of a cyclic selection of several standard values by means of control keys (the arrow keys, etc.). In cases where it is possible, constants are specified directly, with the necessary protection against incorrect values. The entire control of the input and output is implemented through an EMT processor; diagnostics of the hardware and error messages are controlled through a TRAP processor.

Most of the data processing time is taken up by the FFT. We implemented a standard Cooley-Tukey algorithm in integer arithmetic. The speed of this algorithm is equal to the speed of the FFT program of [1]. There is the difference that, in order to economize on memory, the sine table is converted for each new value of the length of the data file to be processed, and a complex repacking algorithm is used to accelerate the calculations of the FFT of a real vector. The operating time of the algorithm (the transformation of 1024 real points) on various software-compatible computers is shown in table 1. Interestingly, the speed of a single-board microcomputer such as the MS1201.02 is only 10 percent lower than that of an Elektronika 100/25 minicomputer, and it is 20 percent better than the speed of the Elektronika 81 or Elektronika 60.1 microcomputer.

The program described above has made it possible to implement a signal analyzer which is controlled as an ordinary instrument. After the power is turned on, the program is automatically loaded, and a 10-item menu is displayed:

0. Set parameters (data acquisition)
1. Program 1 (display input data)
2. Program 2 (power spectrum)
3. Program 3 (initial data and spectrum)
4. Control processing
5. Control cursor
6. Generate hard copy (on a D-100 printer)
7. Control display
8. Plan memory
9. Edit program

Each instruction has its own submenu, or it synthesizes information about the current functions of the control keys.

Program 2, typical of spectral analysis, includes the acquisition of a data file (usually, 512 readings), a multiplication of this file by a Hamming window, the calculation of FFT of the resulting file, the generation of spectral amplitudes, the conversion of these amplitudes to logarithmic scale, the normalization to a given base level, and transmission to a graphics field on the screen with a given dynamic range and an emphasis of up to 10 dB. The resulting periodograms can be averaged in a linear or a logarithmic regime, and the type of window can be changed (rectangular, triangular, Henning, Hamming or user-defined). After a sufficient buildup of the spectrum, one can move a cursor over the graphics window and specify in digital form the relative amplitude of the spectrum at the frequency selected by the cursor. After instruction 6 is selected, the printer generates a copy of the image along with data specified by the cursor and a complete set of values of the parameters of the given measurement. The program takes about 10 Kwords of memory and allocates the remaining 18 Kwords for user data. It thus becomes possible to generate up to 4096 lines in a power spectrum (without a regime of averaging, and with a rectangular or Henning window).

Finally, table 2 summarizes the characteristics of certain common types of signal analyzers; the bottom row shows the characteristics of the new device which we have been discussing here. Although the input characteristics and the processing rate are slightly lower than those of the foreign samples, we believe that the open nature of this system and the possibility of adding to the complexity of the subsequent processing completely offset that disadvantage, particularly in view of the many problems to be solved in the low-frequency and infralow-frequency regions. Furthermore, the input characteristics of this apparatus could be improved significantly by using other signal input modules. The signal band which can be processed in real time could be increased to 25-30 kHz by using fast peripheral processors of the type described in [2].

Bibliography

1. LABORATORY SUBROUTINES MANUAL, Order N AA-C984-TC, Maynard, MA, DEC, 1978.

2. Tolstykh, B.L., Talov, I.L., Plotnikov, V.V., and Bondarovich, G.G., "Elektronika MT-70 Fast Peripheral Processor," UPRAVLYAYSHCHIYE SISTEMY I MASHINY, No 4, 1983.

UDC 681.3

'ADIMEX AGRO' ADAPTIVE IMITATIVE EXPERT SYSTEM FOR TECHNOLOGICAL PROCESS CONTROL IN FARMING

907G0045A Kiev MEKHANIZATSIYA I AVTOMATIZATSIYA UPRAVLENIYA: NAUCHNO-PROIZVODSTVENNYY SBORNIK in Russian No 4, Oct-Dec 89, pp 38-41 [MS Received 11 May 89]

[Article by A.A. Shevchenko, candidate of agricultural sciences, and V.G. Lapa, candidate of technical sciences]

[Text] Large systems often encompass objects of control which, in principle, cannot be studied in active experiment (action-reaction) modes. Such systems and objects include chemical and nuclear reactors, large enterprises and associations, economic and ecological regional and global systems, and objects in the agroindustrial complex.

In solving problems in the ecology, economics, or management, forecasting is of primary importance. The consequences of any decision or purposeful action should be taken into account as accurately as possible. The volume and completeness of information about an object, its measured and unmeasured characteristics, and the state of and changes in the environment stipulate different approaches to solving a problem. The fundamental concept is based on three approaches--deterministic, probability and adaptive.

The deterministic approach is based on known data or data that can be determined beforehand with the requisite accuracy.

The probability approach takes into account the influence of obstructing parameters (noises, interference). The a priori characteristics are considered probable and defined according to known implementations.

Both approaches are based on a large volume of selected observation data, i.e., representative statistics are required.

The adaptive approach is unavoidable when solving problems, for which the initial data are contained in brief samplings. Often, not only are the mathematical descriptions of objects and of the environment unknown beforehand, but, for a number of reasons, it also seems impossible to determine them. The quality of a problem's solution should improve in proportion to the accumulation of information. The system should be capable of self-organization and forecasting [1].

Study of the objects during the process of their natural functioning (passive experimentation) is characterized by a substantial growth in the volume of data and requires the application of fundamentally new technologies for collecting, analyzing, sampling and processing of information. The improvement of technical equipment (universal and personal computers) makes it possible to organize and centralize information streams in automated information retrieval systems, automated data processing systems, and automated management systems (ASU), which either exist or are being developed. At the present time, the main trend is to develop universal technologies for information processing, subsequently adapting them to task-oriented systems. However, intensification in the area of automatic control problems requires the creation of specialized information technologies, taking into account the features of specific ASU.

The "Agroprognoz" Automated Management System (developed by the Ukrainian branch of "AIUS-Agroresursy" VNITs) is intended for on-line control of production processes in farming. Models for forecasting the phenological phases of plant development, which considers the existing and predicted agrometeorological situation, forms a basis for development of an ASU. Recommendations are being drafted for the application of technological methods, aimed at achieving maximum crop yield. The "Agroprognoz" ASU includes the automated information subsystems (AIP): "Pogoda" [Weather]; "Fenologiya" [Phenology]; and the "Zashchita Rasteniy" [Plant Protection] and "Tekhnologiya" [Technology] information management subsystems (IUP) [2].

Input information for the system: agrochemical, agrometeorological, plant growth and development rates, plant biometric indicators, quantitative indicators for productivity factors, information on the most common pests and diseases, their characteristics and thresholds of destructiveness, characteristic ways of protection, information on the phytosanitary condition of crops, on soils in natural climatic zones and provinces, specific features of regionalized agricultural crop varieties, a description of cultivation methods, and ratings of the effectiveness of technological methods used under various soil and agroclimatic conditions.

Output information: characteristics of agrometeorological conditions for the periods of crop growth and development, assessment of weather conditions for current vegetation, a prediction of basic agrometeorological factors, time periods for start of phenophase and biometric indicators on the prehistory and for the current vegetation period, an evaluation and forecast of crop conditions for a requested date, prediction of time periods for appearance of diseases and spread of pests, information on recommended means of protection, recommendations on most effective agrotechnical methods to use in a situation that has specifically occurred or is predicted.

The enumerated data and knowledge structures can conditionally be divided into three basic groups [3]: established information (supplied by state services), special and normative information (biological and genetic features of crops, the corresponding constants, technical and economic indicators and reference materials, data from scientific research and scientific production organizations, from zone varietal testing stations, and from special services organized on farms) and information on the material and technical possibilities of farms, their plan and economic indicators, the distribution structure for areas being planted, etc.

In the process of developing the "Agroprognoz" ASU, new mathematical (multi-factor regression and imitative) models of plant cultivation production processes are developed, and verification is done for existing ones. However, if sufficiently representative statistical material has accumulated for the information streams and AIP models, data on the conditions and results of utilization of means of protection and agrotechnical methods for the IUP are contained, as a rule, in short samples from the results of observations and are a component part of the knowledge of expert specialists. The problem of creating the system has made it necessary to develop specifications for a technology for processing the results of meteorological, phenological and biological observations and for modeling biological objects. In this regard, the methods and approaches being developed within the framework of "artificial intelligence" were used, and the ADIMEX AGRO adaptive imitative expert management system was created as a result.

In the computerized ADIMEX system, knowledge is represented by the sum total of frames (situations, scenarios) and deduction rules. The original set of frames is determined by experts in the field (in this case, by agronomists, soil scientists, agrometeorologists, specialists in plant protection, and others).

The expert "supplies" knowledge to the engineering and systems unit (interpreter), which knows the structure of ADIMEX and is capable of representing knowledge. Knowledge is often "fragments of prose," containing rules of the type: "if..., then...," "if..., then often...," "if..., then with great probability...," etc. Fairly complex statements may take the place of the dots, containing both numbers, as well as linguistic modifiers of the type: "big," "average," and "rarely," as well as vague concepts, for example, "with great probability," "surface plowing," and "harvest shortfall." Knowledge is then stored in the computer in special logical structures and processed through methods, known as "knowledge engineering." User access to the memory of a large computer at his work place is provided by a display, connected to the computer via the telephone network or a special cable. Considering the rates and prospects for computer hardware development, it will be possible in the near future to create large imitative expert systems based on personal computers.

From time to time, ADIMEX is "reviewed" by expert verifiers. These may be specialists, invited from outside, or the experts themselves, who formed this system, can also perform this role. From the initial set of frames, a knowledge base [KB] is formed, which is used for comparison with existing and predicted situations for the purpose of drafting recommendations and management actions. During operation, the KB is continuously supplemented and corrected as a result of receipt of new information on the outcomes of work by the system itself, as well as information from literary sources, reports, and developments--this is the adaptive systems operation mode (continuous updating). ADIMEX should not only make decisions in a timely manner, but also give an acceptable explanation as to why precisely this, and not some another method should be chosen, and it should rate (for instance, in points) solution variants for different situations from the viewpoint of economics, ecology, production quality, implementation complexity, etc.

In order to represent knowledge, structures are used, based on the calculation of mathematical logic statements and predicates, for example, representations of the implication type, IF--THEN: condition--action. Part of the IF rule is the formulation of a number of conditions (prerequisites), under which the rule is applicable:

```
IF   1)... and... or...;
     2)... or...;
     3)...;
THEN 1)... and...;
     2)..., but not...;
     3)... or...
```

The effect of the "$\overline{\text{or}}$" part of a THEN rule is a conclusion, which must be made if the condition is met.

The ALSIR algorithmic language for describing situations and solutions (recommendations) is structured on the laws for calculating statements and predicates. For example, the block "situation--solution" can be presented in the following form: ('zone' $Z_1 \wedge$'soil' $P_2 \wedge$, 'precursor' R_4 ...) $\longrightarrow$ ('technological methods' $T_2 \wedge$, 'aggregate' $A_3 \vee A_1 \wedge$'fertilizer' F_5 ...), where $\wedge$ is the sign AND--conjunction; $\vee$ is the OR sign--disjunction; and $\longrightarrow$ is the IF-THEN sign--implication. Logical relations tables described in an algorithmic language are programmed and used in order to form knowledge bases.

The formation and management of knowledge structures on a computer is supported by a knowledge base management system, based on the relational model of knowledge representation. Relational knowledge bases are sufficiently user-friendly for those who have little experience with computers.

Raising the immediacy of control, multi-variant solutions with ecological and economic assessments, optimization of agrotechnical methods, and rational use of material and labor resources are the basic factors which ensure the economic expediency of developing and introducing an adaptive imitative expert system.

Practical use of the "Agroprognoz" ASU makes it possible to increase the production output per hectare of plowed field by 0.2-0.3 centners in a grain equivalent, or by 2.5 rubles, which, taken for the UKSSR on the whole, amounts to over 75 billion rubles. Taking into account development outlays, the expected economic effect from using the system in the republic is about 40 million rubles.

BIBLIOGRAPHY

1. Lapa, V.G. "Metody Predskazaniya i Predskazyvayushchiye Sistemy" [Prediction Methods and Forecasting Systems]. Kiev, 1980.

2. Shevchenko, A.A., and Lapa, V.G. "Automated Management of Production Processes in Plant Cultivation." ZEMLEDELIYE, No 4, 1989.

3. Yakushev, V.P., Belokoskov, A.V., and Lomakin, V.S. "Expert System for Agrotechnical Decision Support in Crop Planning." VESTNIK SELSKOKHOZYAYSTVENNOY NAUKI, No 4, 1984.

UDC517.444;621.391.1;681.3.012

IMPLEMENTATION OF PARALLEL DIGITAL-FILTERING ALGORITHMS ON THE PS-200 MULTIPROCESSOR COMPUTATION COMPLEX

18630180A Novosibirsk AVTOMETRIYA in Russian No 6, Nov–Dec 88 (manuscript received 16 Jul 87) pp 3–9

[Article by O.Ye. Baklar ova, M.V. Zyuzin and A.V. Lyulyakov, Novosibirsk: "Implementation of Parallel Digital-Filtering Algorithms on the PS-200 Multiprocessor Computation Complex," published under the rubric "Real-Time Computation Systems"]

[Text] Digital-filtering methods are very popular in the processing of experimental data. One reason is the widespread dissemination of fast parallel processing computers. A second is the appearance of new and more efficient algorithms for implementing digital filters, which are, in turn, easily adaptable to parallel processing.

In this paper we describe some algorithms for the digital filtering of signals and images which have been realized on the PS-2000 multiprocessor computation complex [MVK]. We report some calculations which demonstrate the efficiency of the processing of experimental data.

Brief Description of the PS-2000

In order to exploit the advantages of a microprocessor system it is necessary to consider the functional capabilities of its hardware facilities. Let us examine some features of the PS-2000 [1].

The PS-2000 complex falls in the category of single-instruction-stream, multiple-data-stream [SIMD] computers. It includes an SM-2 computer, which controls the system, and a PS-2000 matrix processor, which performs the basic data processing. The matrix processor can contain up to 64 uniform processor elements [PE], which act upon instructions from the control device [UU]. Each processor element has its own working memory [OP] of 26K 24-bit words, and an arithmetic-and-logic unit [ALU], which contains 16 general-purpose registers that perform rapid data processing. The exchange of data between

Table 1.

Instruction	Number of Instructions, s^{-1}			Instruction	Number of Instructions, s^{-1}		
	1 PE	16 PE	64 PE		1 PE	16 PE	64 PE
$+_{x}$	10^6	$16\cdot10^6$	$64\cdot10^6$	$*_{x}$	$0,4\cdot10^6$	$6,4\cdot10^6$	$25,6\cdot10^6$
$+_{l}$	$0,8\cdot10^6$	$12,8\cdot10^6$	$51,2\cdot10^6$	$*_{l}$	$0,3\cdot10^6$	$4,8\cdot10^6$	$19,2\cdot10^6$

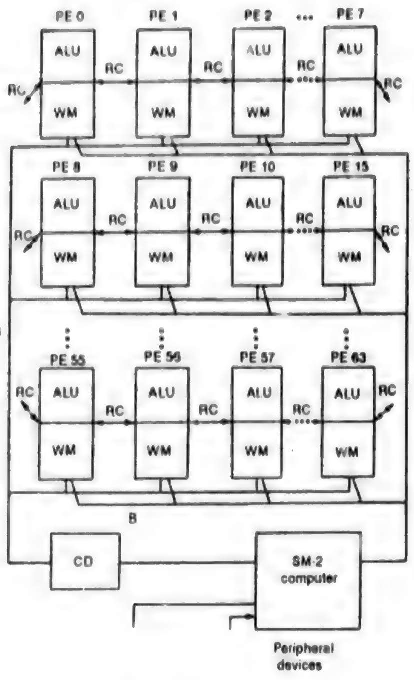

Figure 1.

neighboring processor elements is transacted along regular channels [RK], while the exchange of data between a processor element and the control element is transacted along buses [MK]. Figure 1 is a simplified schematic diagram of the PS-2000 complex. The number p of processor elements operating in the system can be a multiple of eight; i.e., p = 8, 16, …, 64. All of these elements synchronously perform computational operations on data stored in the general-purpose registers or in the memory layers of the processor elements. Each register and memory layer consists of a set of p 24-bit words. Operations with vectors of length p are carried out on a component basis and simultaneously:

$$(c_1, \ldots, c_p) = (a_1, \ldots, a_p) * (b_1, \ldots, b_p), \quad c_i = a_i * b_i, \quad i = 1, \ldots, p,$$

where the asterisk (*) is some arithmetic or logic operation. Various shifts of the binary digits of the words can be used to accelerate multiplication operations. In the exchange of data between processor elements the operation of cyclic permutation is used; the components of a vector are moved right or left. For example, a cyclic permutation of the vector $(a_1, \dots, a_p)$ three words to the right generates a vector $(a_{p-2}, a_{p-1}, a_p, a_1, a_2, \dots, a_{p-3})$. In the execution of certain instructions, only some of the processor elements may be activated. The others will be in a wait state. In addition to the vector working memory of the processor elements (M) there is a memory (H) which is used for scalar instructions and constants.

Table 1 shows some characteristics of the speed of the system in the execution of basic arithmetic instructions. Here, $+_f$ and $+_p$ are fixed-point and floating-point summation instructions, while $*_f$ and $*_p$ are corresponding multiplication instructions.

In the use of the PS-2000 complex, it is desirable to adhere to the following recommendations: 1) the computation algorithm should be in vector form, and the length of the vectors should be proportional to the number of processor elements; 2) data which are used frequently should be stored in registers; 3) computational instructions, memory accessing and data input and output should be alternated, since these actions can be carried out in parallel; and 4) division instructions should be eliminated from the algorithm to the greatest possible extent.

Real-Time Signal Processing

In the processing of signals (acoustic, radar, etc.), digital filters often must work in real time. Let us assume that signal readings f(n) are arriving continuously at a certain frequency ω, ..., f(n - 1), f(n), f(n + 1), ... The task is to apply to f(n) the filter

$$g(n) = \sum_{l=0}^{N} k(l)\, f(n-l), \qquad n = 0, 1, \dots, \tag{1}$$

with a given pulsed characteristic k(n) and to generate a sequence of readings g(n) at the same frequency.

Let us examine some algorithms for the real-time processing of a signal f(n) by filter (1) which are implemented on the PS-2000.

We can write expression (1) in an equivalent form:

$$g(n) = \sum_{l=0}^{N} k_1(l)\, f(n_1+l), \qquad n = 0, 1, \dots,$$

where $k_1(l) = k(N - l)$; $l = 0, \dots, N$; $n_1 = n - N$.

Let us assume that the number of coefficients in pulsed characteristic $k_1(l)$ is a multiple of the number p, $N + 1 = pc$, where $1 < c < 7$. We denote by R_i ($i = 1, \dots, 16$) the general-purpose registers, while S_i ($i = 1, \dots, 16K$) are the layers of the working memory of the processor element. We will use the following instructions:

1) summation, $R_i + R_j$, and multiplication, $R_i * R_j$;
2) writing of the value of R_i in R_j, $R_i \rightarrow R_j$;
3) shifting of words in a register (cyclic permutation);
4) writing a layer of memory in a register, $S_i \rightarrow R_j$;
5) sending one word from R_i to R_j;
6) input and output of one word from a register.

The data processing can be carried out in either the integer-number format (for this purpose, the real coefficients $k_1(n)$ are written in the form $m(n)/2^M$, where M and m(n) are integers, with $n = 0, \dots, N$) or the real-number format. The integers are subsequently converted into real numbers.

Let us assume that each reading is processed by all of the processor elements. We store the coefficients $k_1(0), \dots, k_1(N)$ and the readings $f(-N), \dots, f(0)$ in registers $R_1, \dots, R_c$ and $R_{c+1}, \dots, R_{2c}$, respectively. To calculate the value of g(0), g(1), … we use

```
Algorithm 1:
1. $n = 0$.
2. $0 \rightarrow R_{16}$,  $i = 1, j = c + 1, l = 1, q = 1$.
3. $R_i$ (coefficients) $*$ $R_j$ (readings) $+ R_{16} \rightarrow R_{16}$.
4. If $1 < c$, then word 1 from $R_{j+1}$, else $f(n + 1) \rightarrow$ word 1 of $R_j$.
5. Shift of $R_j$ one word to the left.
6. $i = i + 1$, $j = j + 1$ if $1 \le c$ in step 3.
7. $R_{16}$ + (result of a leftward shift of $l$ words in $R_{16}$) $\rightarrow R_{16}$.
8. If $q < \log_2 p$, then $l = 2l$, $q = q + 1$ in step 7; otherwise, send g(n) from word 1 in $R_{16}$
9. $n = n + 1$ in step 2.
```

If a filter with a large number of coefficients is to be realized, it becomes necessary to use the memory of the processor elements. There will accordingly be large-scale exchanges between the memory and the registers and thus an increase in processing time.

The second algorithm is essentially one in which each processor element processes its own reading. We introduce groups of readings

$$G(m) = \{g(n), \dots, g(n + p - 1)\} \quad \text{and} \quad F(m) = \{f(n - N), \dots, f(n - N + p - 1)\},$$
$$n = mp, m = 0, 1, \dots$$

To calculate the values of group G(m) with the help of F(m), …, F(m + c) we use

Algorithm 2:

```
1. $0 \rightarrow R_1$,  $l = 0$,  $F(m) \rightarrow R_3$.
2. $k_1(l) \rightarrow R_2$.
3. $R_1 + R_2 * R_3 \rightarrow R_1$.
4. If $l = N$, then branch to step 7.
5. $l = l + 1$; in word 1 of register $R_3$ we store $f(n - N + p + l)$.
6. Shift $R_3$ one word to the left and branch to step 2.
7. Extract group G(m) from $R_1$.
```

The processing of G(m + 1) requires the repeated use of F(m), …, F(m + c) and of the new group (Fm + c + 1). The storage of the results should be organized in the available registers. The coefficients $k_1(l)$ are stored in the first N + 1 layers of S_{l+1} (l = 0, …, N) of the working memory. Each layer of S_{l+1} contains p coefficients $k_1(l)$.

We will assume below that the signal arrives at a frequency ω = 48 kHz. This figure corresponds to the accepted standard in the development of devices for the digital processing of studio-link acoustic signals. The algorithms which have been proposed have been implemented in the language HPS for the PS-2000 processor [1]. Numerical experiments on real-time signal processing were carried out in a system containing 16 processor elements. It was established as a result that algorithm 1 makes it possible to realize a filter which operates in real time with no more than 16 coefficients. For algorithm 2, the calculations were carried out with both integer and real numbers. The results are summarized in table 2. We see from this table that a filter with 48 coefficients can cope with the processing of signals arriving at a rate of 48 kHz on the PS-2000 with 16 processor elements.

Implementation of Image-Processing Algorithms

A nonrecursive filter which acts on an image $f(n_1, n_2)$ is defined by a convolution transformation:

$$g(n_1, n_2) = k \widehat{*} f(n_1, n_2) = \sum_{l_1, l_2 = -\infty}^{\infty} k(l_1, l_2) f(n_1 + l_1, n_2 + l_2), \qquad (2)$$

where $k(n_1, n_2)$ is the pulsed characteristic of the filter. We denote by $K(\xi_1, \xi_3)$ the frequency characteristic:

$$k \widehat{*} e^{i(\xi_1 n_1 + \xi_2 n_2)} = K(\xi_1, \xi_2) e^{i(\xi_1 n_1 + \xi_2 n_2)}.$$

The usual task is to process an image $f(n_1, n_2)$ by a digital filter with some pulsed characteristic $k(n_1, n_2)$ which corresponds to a given function $K(\xi_1, \xi_2)$.

Table 2.

Number of readings to be processed	Number of coefficients	Required time, sec	Expended time, sec	
			Integer format	Real format
708 000	32	16	12,35	12,50
1 024 000	32	21	10,36	10,07
708 000	48	10	15,40	—
1 152 000	48	24	23,10	—
1 536 000	48	32	30,80	—

An effective operation of a combinational assembly is frequently used in calculations on a wide class of two-dimensional digital filters (bandpass filters, rejection filters, inverse filters, etc.) [2-4]. In this case the pulsed characteristic is written in the form

$$k_1(n_1, n_2) = \sum_{j=0}^{N} b_j k_0^{*j}(n_1, n_2). \tag{3}$$

Here, b_j are the transformation coefficients; $k_0^{*j}(n_1,n_2)$ is a convolution of a function $k_0(n_1,n_2)$ with itself j times (this function has a small number of coefficients; for example, it could be lumped at 3 × 3 points of a grid). A decomposition of this sort of the function $k_1(n_1,n_2)$ makes it possible to significantly reduce the number of computational operations during the operation of a digital filter.

Let us describe some algorithms for processing images with digital filters with pulsed characteristics $k(n_1,n_2)$ and $k_1(n_1,n_2)$ which have been implemented on the PS-2000 multiprocesssor computation complex. We assume that the values of the image $f(n_1,n_2)$ (n_1 = 1, ..., M1; n_2 = 1, ..., M2) are put in a matrix A(M1,M2), the numbers of rows and columns of which are multiples of the number of processor elements p: M1 = K1p, M2 = K2p. We assume that the pulsed characteristic $k(n_1,n_2)$ spans N1 × N2 nodes of the mesh: $n_1 = -\bar{n}_1, \ldots, \bar{n}_1$, $n_2 = -\bar{n}_2, \ldots, \bar{n}_2$, $N1 = 2\bar{n}_1 + 1$, $N2 = 2\bar{n}_2 + 1$. We are to calculate

$$g(n_1, n_2) = \sum_{l_1=-\bar{n}_1}^{\bar{n}_1} \sum_{l_2=-\bar{n}_2}^{\bar{n}_2} k(l_1, l_2) f(n_1 + l_1, n_2 + l_2) \tag{4}$$

at the points $n_1 = 1 + \bar{n}_1, \ldots, M1 - \bar{n}_1$; $n_2 = 1 + \bar{n}_2, \ldots M2 - \bar{n}_2$.

We write the matrix A in the memory of processor e..ements M. In the memory of the first processor we store in succession the rows with numbers 1, ..., K1 + N1 - 1; in that of the second processor we store the rows beginning from K1 + 1 through 2K1 + N1 - 1; etc. Processor J will contain rows (J - 1)K1 + 1, ..., J, K1 + N1 - 1 (J = 1, ..., p).

The coefficients of the pulsed characteristic $k(n_1,n_2)$ are written in succession in scalar memory H in the form of a one-dimensional vector B of length N1 × N2:

$$B(i) = k(n_1,n_2), \quad n_1 = -\bar{n}_1, \ldots, \bar{n}_1, \quad n_2 = -\bar{n}_2, \ldots, \bar{n}_2,$$
$$i = (n_1 + \bar{n}_1)N2 + n_2 + \bar{n}_2 + 1.$$

We multiply the first N2 layers of the memory by the elements 1, …, N2 of vector B, and we sum them in register B1. Layers with numbers M2 + 1, …, M2 + N2 are multiplied by the next coefficients, B(B2 + 1), …, B(2·N2), and they are also summed with R1. We carry out corresponding operations with layers 2KM + 1, …, 2KM + M2, K = 2, …, M1 - 1. As a result, register R1 obtains the values of the image $g(n_1,n_2)$ with indices $n_1 = (J - 1)K1 + \bar{n}_1 + 1$, $J = 1, \ldots, p$, $n_2 = \bar{n}_2 + 1$.

In the memory of processor elements M we carry out a shift one layer downward, and we repeat the operations described above. In this manner we calculate the elements $g(n_1,n_2)$, with indices $n_1 = (J - 1)K1 + \bar{n}_1 + 1$, $J = 1, \ldots, p$, $n_2 = \bar{n}_2 + 2$.

Repeating this sequence of actions in a cycle M1 × M2/p times, we obtain the other values of the image $g(n_1,n_2)$ (see (4)), and we complete the processing. This algorithm obviously allows full exploitation of the processor elements.

On the basis of a convolution operation, we achieve a parallel-processing implementation of a filter with a pulsed characteristic $k_1(n_1,n_2)$, calculated by the operation of a combinational assembly (3). For this purpose, it is necessary to carry out a convolution of the image $f(n_1,n_2)$ with the function $k_0(n_1,n_2)$ repeatedly in a separate part of working memory M. The first M1 × M2/p layers of memory are used to store the result.

Formally, the working algorithm of filter (4) can be written as follows:

```
1. $n = 1$.
2. $0 \rightarrow R1$,  $l = 0$,  $k = n - 1$,  $i = 1$.
3. $f = 1$.
4. $R1 + S(k + j) * B(l + j) \rightarrow R1$.
5. $j = j + 1$,  if  $j \leq N2$ in step 4.
6. $i = i + 1$,  if  $i > N1$ in step 8.
7. $l = l + N2$,  $k = k + M2$ in step 3.
8. $R1 \rightarrow S(n)$,  $n = n + 1$ in step 2.
```

Example. Figure 2 shows an image $g(n_1,n_2)$ containing 275 × 512 readings, which has been distorted by a convolution operation as in (2) with an instrumental function

$$k(n_1,n_2) = k_1 * k_1(n_1,n_2),$$

where $k_1(n_1,n_2)$ was obtained through a McClellan transformation [2, 4] of the one-dimensional step function

$$k_0(n) = \begin{cases} 1/5, & n = -2, \ldots, 2; \\ 0 & \text{иначе}. \end{cases}$$

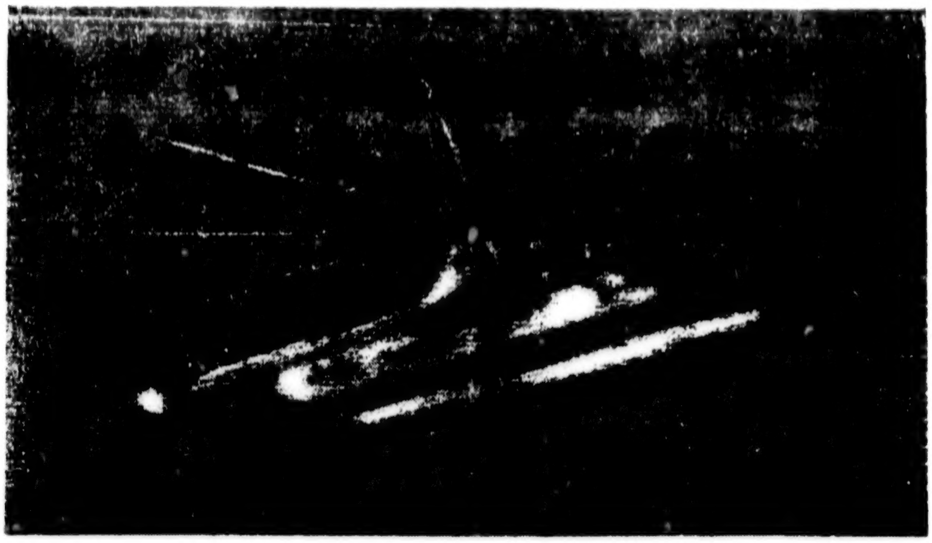

Figure 2.

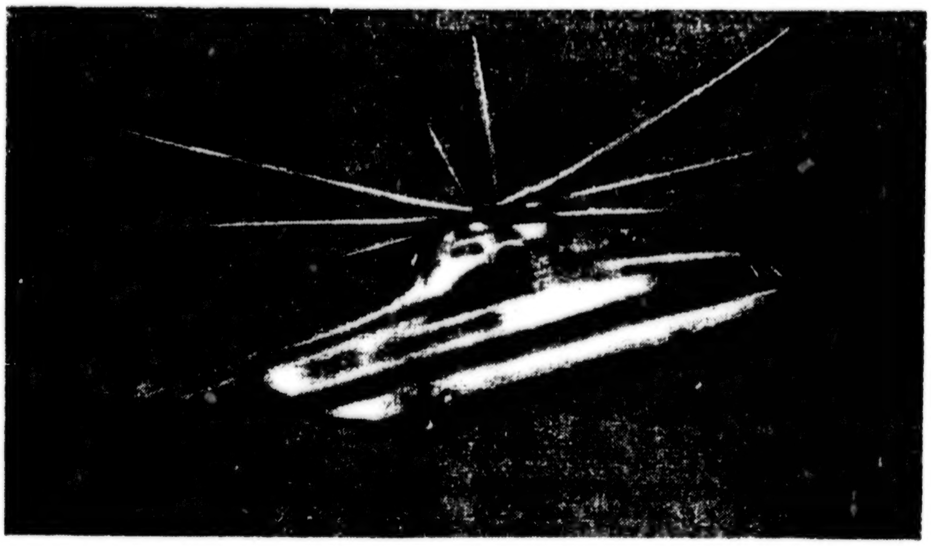

Figure 3.

The function $k_1(n_1,n_2)$ is represented by the matrix

$$\begin{pmatrix} 0{,}0125 & 0{,}05 & 0{,}075 & 0{,}05 & 0{,}0125 \\ 0{,}05 & 0{,}05 & 0 & 0{,}05 & 0{,}05 \\ 0{,}075 & 0 & 0{,}05 & 0 & 0{,}075 \\ 0{,}05 & 0{,}05 & 0 & 0{,}05 & 0{,}05 \\ 0{,}0125 & 0{,}05 & 0{,}075 & 0{,}05 & 0{,}0125 \end{pmatrix}.$$

The frequency characteristic $K_1(\xi_1,\xi_2)$, which corresponds to $k_1(n_1,n_2)$, is of varying sign, so the problem of reconstructing the image $f(n_1,n_2)$ is an ill-posed problem. As a first step, we amplify the high frequencies by using the filter $\tilde{g}(n_1,n_2) = \dot{q} * g(n_1,n_2)$ with the pulsed characteristic

$$q(n_1, n_2) = \begin{pmatrix} -\frac{3}{2} & -3 & -\frac{3}{2} \\ -3 & 19 & -3 \\ -\frac{3}{2} & -3 & -\frac{3}{2} \end{pmatrix}.$$

To calculate the pulsed characteristic $\tilde{k}(n_1,n_2)$ of the inverse filter we use the combinational assembly operation [3]:

$$\tilde{k}(n_1, n_2) = \sum_{j=0}^{3} b_j (q * k(n_1, n_2))^{*j},$$

where $b_0 = 4$, $b_1 = -6$, $b_2 = 4$, $b_3 = -1$.

Figure 3 shows the reconstructed image $\tilde{f}(n_1,n_2) = \tilde{k} * \tilde{g}(n_1,n_2)$. The time required to process this image on the PS-2000 with 16 processor elements was 5 s. By way of comparison, a corresponding processing on an ES-1061 computer took 39.5 s.

Bibliography

1. Shkolnik, K.M., "The PS-2000 Technological Support in the OBRAZ System," in: TEORETICHESKIYE I PRIKLADNYE VOPROSY PARALLELNOY OBRABOTKI INFORMATSII [Theoretical and Applied Questions of Parallel Data Processing], Novosibirsk, VTs SO AN SSSR, 1984.

2. McClellan, J.H., "The design of two-dimensional digital filters by transformations," in: PROCEEDINGS OF THE SEVENTH ANNUAL PRINCETON CONFERENCE ON INFORMATION SCIENCES AND SYSTEMS, , p 247, 1973.

3. Zyuzin, M.V., "Algorithm for Assembling Inverse Filters," AVTOMETRIYA, No 4, 1987.

4. Zyuzin, M.V., "Economical Method for Assembling Two-Dimensional Digital Filters in Image Processing Problems," in: VARIATSIONNO-RAZNOSTNYE METODY V ZADACHAKH CHISLENNOGO ANALIZA [Variational-Difference Methods in Numerical-Analysis Problems], Novosibirsk, VTs SO AN SSSR, 1987.

- END -